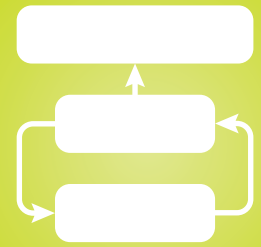


# msdn

magazine



Take the Pain Out  
of Data Binding.....14

## Be Unstoppable

Create high-impact Windows® user experiences  
with DevExpress

**UI CONTROLS  
FOR WPF & WINFORMS**

Free 30-day trial  
@ [DevExpress.com/try](http://DevExpress.com/try)



**"A must have for any serious developer."**

Ron Lindsey

**"DevExpress works as a major  
workforce multiplier."**

Peter Van Zyl

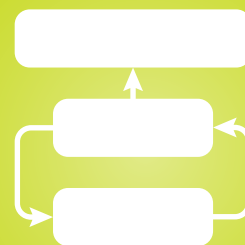
**"One of the most powerful, feature rich  
control suites on the market."**

Chris Todd

**#UseTheBest**

# msdn

magazine



Take the Pain Out  
of Data Binding.....14

A Better Way to Implement  
Data Binding in .NET  
**Mark Sowul** .....14

Working with Local Databases  
in Xamarin.Forms Using SQLite  
**Alessandro Del Sole** .....24

Leverage CQRS to Create  
Highly Responsive Systems  
**Peter Vogel**.....38

Applying AI to a Multi-Agent  
Mini-Basketball Game  
**Arnaldo Pérez Castaño** .....48

## COLUMNS

### CUTTING EDGE

Reflections on Code First,  
Persistence and  
Domain Modeling  
Dino Esposito, page 6

### DATA POINTS

The New Azure DocumentDB  
Node.js SDK  
Julie Lerman, page10

### TEST RUN

Matrix Inversion Using C#  
James McCaffrey, page 58

### THE WORKING PROGRAMMER

How To Be MEAN: Let's Be DEAN  
Ted Neward, page 62

### ESSENTIAL .NET

Visual Studio 2015 with  
.NET Core Tooling  
Mark Michaelis, page 70

### MODERN APPS

Build a Wi-Fi Scanner  
in the UWP  
Frank La Vigne, page 76

### DON'T GET ME STARTED

Missing the Target  
David S. Platt, page 80

# ASP.NET MVC REPORTING

The first cross-browser, true  
WYSIWYG HTML5-based  
document editor.

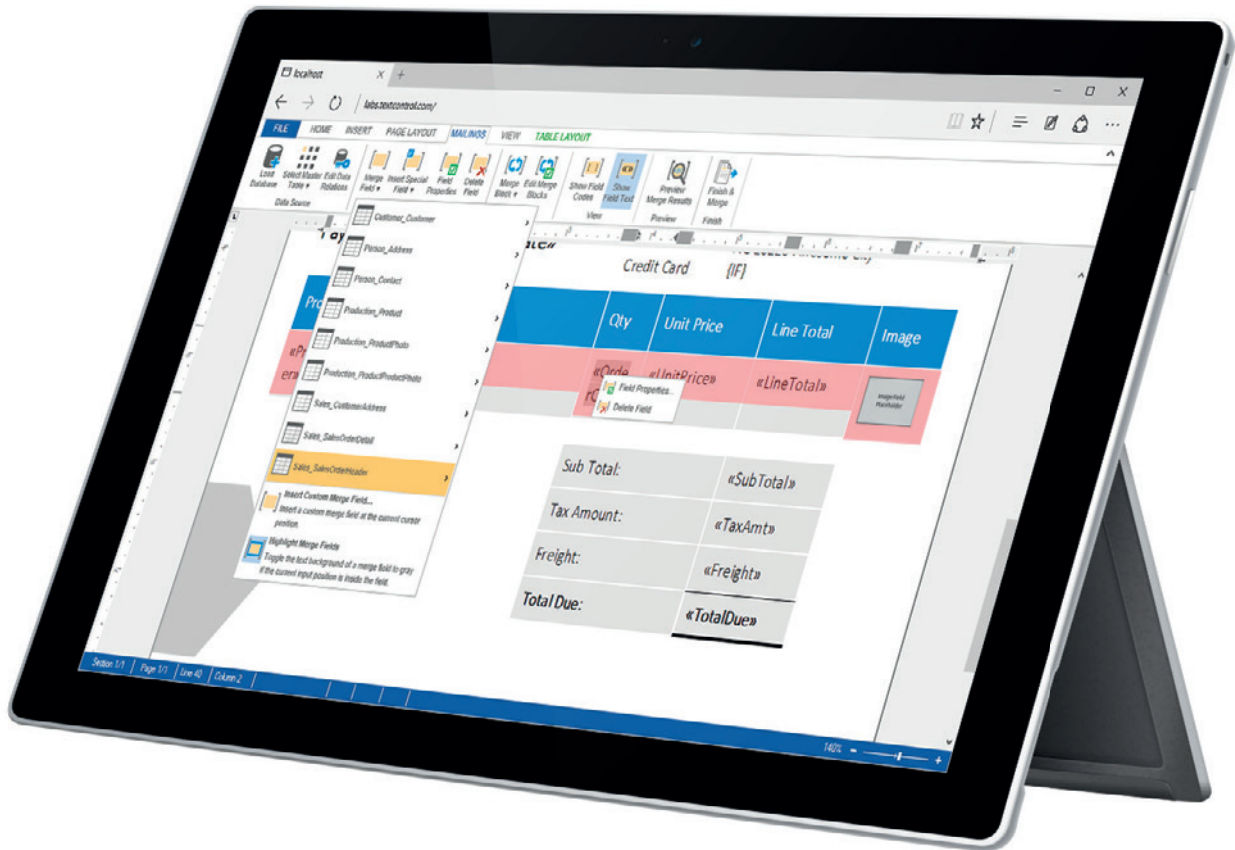
Now with full  
ASP.NET MVC support.



Give your users an MS Word compatible editor to create powerful documents and reporting templates anywhere - in any browser. Feature-complete including mail merge, sections, headers and footers, drawings and shapes, tables, barcodes and charts.

Available for ASP.NET Web Forms and MVC.





Edit and create  
MS Word  
documents



Create and modify  
Adobe® PDF  
documents



Create reports  
and mail merge  
templates



Integrate with  
Microsoft®  
Visual Studio

PM> Install-Package TXTextControl.Web

Live demo and 30-day trial version download at:  
[www.textcontrol.com/html5](http://www.textcontrol.com/html5)

**X13**  
released

Software ▪ Training ▪ Consulting

**TEXT CONTROL**

**General Manager** Jeff Sandquist

**Director** Dan Fernandez

**Editorial Director** Mohammad Al-Sabt [mmeditor@microsoft.com](mailto:mmeditor@microsoft.com)

**Site Manager** Kent Sharkey

**Editorial Director, Enterprise Computing Group** Scott Bekker

**Editor in Chief** Michael Desmond

**Features Editor** Sharon Terdeman

**Features Editor** Ed Zintel

**Group Managing Editor** Wendy Hernandez

**Senior Contributing Editor** Dr. James McCaffrey

**Contributing Editors** Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt, Bruno Terkaly

**Vice President, Art and Brand Design** Scott Shultz

**Art Director** Joshua Gould



**President**  
Henry Allain

**Chief Revenue Officer**  
Dan LaBianca

**Chief Marketing Officer**  
Carmel McDonagh

## ART STAFF

**Creative Director** Jeffrey Langkau

**Associate Creative Director** Scott Rovin

**Senior Art Director** Deirdre Hoffman

**Art Director** Michele Singh

**Assistant Art Director** Dragutin Cvijanovic

**Graphic Designer** Erin Horlacher

**Senior Graphic Designer** Alan Tao

**Senior Web Designer** Martin Peace

## PRODUCTION STAFF

**Print Production Coordinator** Lee Alexander

## ADVERTISING AND SALES

**Chief Revenue Officer** Dan LaBianca

**Regional Sales Manager** Christopher Kourtoglou

**Account Executive** Caroline Stover

**Advertising Sales Associate** Tanya Egenolf

## ONLINE/DIGITAL MEDIA

**Vice President, Digital Strategy** Becky Nagel

**Senior Site Producer, News** Kurt Mackie

**Senior Site Producer** Gladys Rama

**Site Producer** Chris Paoli

**Site Producer, News** David Ramel

**Director, Site Administration** Shane Lee

**Site Administrator** Biswarup Bhattacharjee

**Front-End Developer** Anya Smolinski

**Junior Front-End Developer** Casey Rysavy

**Executive Producer, New Media** Michael Domingo

**Office Manager & Site Assoc.** James Bowling

## LEAD SERVICES

**Vice President, Lead Services** Michele Imgrund

**Senior Director, Audience Development & Data Procurement** Annette Levee

**Director, Audience Development & Lead Generation Marketing** Irene Fincher

**Director, Client Services & Webinar Production** Tracy Cook

**Director, Lead Generation Marketing** Eric Yoshizuru

**Director, Custom Assets & Client Services** Mallory Bundy

**Senior Program Manager, Client Services & Webinar Production** Chris Flack

**Project Manager, Lead Generation Marketing** Mahal Ramos

**Coordinator, Lead Generation Marketing** Obum Ukabam

## MARKETING

**Chief Marketing Officer** Carmel McDonagh

**Vice President, Marketing** Emily Jacobs

**Senior Manager, Marketing** Christopher Morales

**Marketing Coordinator** Alicia Chew

**Marketing & Editorial Assistant** Dana Friedman

## ENTERPRISE COMPUTING GROUP EVENTS

**Vice President, Events** Brent Sutton

**Senior Director, Operations** Sara Ross

**Senior Director, Event Marketing** Merikay Marzoni

**Events Sponsorship Sales** Danna Vedder

**Senior Manager, Events** Danielle Potts

**Coordinator, Event Marketing** Michelle Cheng

**Coordinator, Event Marketing** Chantelle Wallace



**Chief Executive Officer**  
Rajeev Kapur

**Chief Operating Officer**  
Henry Allain

**Vice President & Chief Financial Officer**  
Michael Rafter

**Chief Technology Officer**  
Erik A. Lindgren

**Executive Vice President**  
Michael J. Valenti

**Chairman of the Board**  
Jeffrey S. Klein

**ID STATEMENT** MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email [MSDNmag@1105service.com](mailto:MSDNmag@1105service.com) or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o MSDN Magazine, 4 Venture, Suite 150, Irvine, CA 92618.

**LEGAL DISCLAIMER** The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

**CORPORATE ADDRESS** 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 [www.1105media.com](http://www.1105media.com)

**MEDIA KITS** Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), [dlabianca@1105media.com](mailto:dlabianca@1105media.com)

**REPRINTS** For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International Phone: 212-221-9595. E-mail: [1105reprints@parsintl.com](mailto:1105reprints@parsintl.com). [www.magreprints.com/QuickQuote.asp](http://www.magreprints.com/QuickQuote.asp)

**LIST RENTAL** This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: [jl@meritdirect.com](mailto:jl@meritdirect.com); Web: [www.meritdirect.com/1105](http://www.meritdirect.com/1105)

## Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at [Redmondmag.com](http://Redmondmag.com).

E-mail: To e-mail any member of the staff, please use the following form: [FirstInitialLastName@1105media.com](mailto:FirstInitialLastName@1105media.com)  
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)  
Telephone 949-265-1520; Fax 949-265-1528  
4 Venture, Suite 150, Irvine, CA 92618

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)  
Telephone 818-814-5200; Fax 818-734-1522  
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311

The opinions expressed within the articles and other contents herein do not necessarily express those of the publisher.





DESKTOP



TABLET



MOBILE



## OCR FOR ANY CLIENT, DEVICE OR SERVER. DONE.

### LEADTOOLS OCR SDK

*Fast, accurate and reliable optical character recognition for use in any application or environment*

*Utilize multiple cores for unparalleled performance*

*Supports multiple text recognition engines including OCR, ICR, MICR, MRZ and MRP*

*Automatically detect, segment and recognize multiple languages on the same document*

.NET   Windows API   Java   WinRT   Linux   iOS   OS X   Android   JavaScript







## Bad Medicine

In my last two columns, I described lessons that large-scale calamities carry for software development teams. Tragic examples abound, from the misapplied training that impacted the response at Three Mile Island in 1979 (“Going Solid,” [msdn.com/magazine/mt703429](http://msdn.com/magazine/mt703429)) to cognitive biases that caused people to sharply underestimate risk in events as diverse as the 2010 Deepwater Horizon oil spill and 2008 global financial meltdown (“Cognitive Bias,” [msdn.com/magazine/mt707522](http://msdn.com/magazine/mt707522)).

Of course, these examples are external to software development. One incident closer to home is the Therac-25, a medical device that delivered radiation treatments to cancer patients. Over the course of 18 months between 1985 and 1987, the Therac-25 delivered major radiation overdoses to six patients in the United States and Canada, killing at least two. The manufacturer, Atomic Energy of Canada Ltd. (AECL), struggled to pinpoint the causes, even as incidents mounted.

She found that AECL was  
overconfident in its software,  
unrealistic in assessment of risk,  
and deficient in defensive design  
elements such as error checking  
and handling.

The Therac-25 could deliver either a low-power electron or X-ray beam, depending on the prescribed treatment. But a race condition in the software made it possible to expose patients to a massive radiation dose, if the operator switched quickly from X-ray mode to electron beam mode before the beam hardware could move into

position. Two patients in Texas died after they were exposed to the unmoderated, high-power electron beam.

A second bug could cause the device to produce a deadly electron beam while the operator aligned the device on the patient. The culprit: A one-byte variable in the software that would increment to a zero value on each 256th pass. If the operator pushed the set button at the instant the variable incremented to zero, the beam would activate unexpectedly. Several others may have been overdosed due to this software flaw.

Remarkably, these code flaws had been present for years in older Therac models (including the similar Therac-20) without incident. The difference: The Therac-20 employed hardware interlocks that physically prevented an overdose from being administered—the machine would simply blow a fuse. But AECL engineers had replaced those interlocks with software on the Therac-25. And the software was not up to the task.

Nancy Levenson, a professor of Engineering Systems at the Massachusetts Institute of Technology, performed a detailed study of the Therac-25 incidents (you can read the PDF yourself at [bit.ly/1QnJ03h](http://bit.ly/1QnJ03h)). She found that AECL was overconfident in its software, unrealistic in assessment of risk, and deficient in defensive design elements such as error checking and handling. Remarkably, just one developer—of unknown credentials, AECL never identified the person—wrote and tested the Therac software, which was written in PDP-11 assembly language. Levenson found that AECL lacked a robust testing program and assumed that software reused from earlier Therac models would be free of flaws.

Therac-25 reads like a lesson in hubris. Unlike earlier models, the Therac-25 relied almost exclusively on software to ensure safe operation. And yet, AECL relied on a single (apparently, unmanaged) coder and aging, reused software in the device. Now 30 years later, it's a lesson worth contemplating.

Visit us at [msdn.microsoft.com/magazine](http://msdn.microsoft.com/magazine). Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: [mmeditor@microsoft.com](mailto:mmeditor@microsoft.com).

© 2016 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at [microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx](http://microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx). Other trademarks or trade names mentioned herein are the property of their respective owners.

*MSDN Magazine* is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.



## Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

NEW  
v5.5

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module



## Complete Suite of Accurate PDF Components

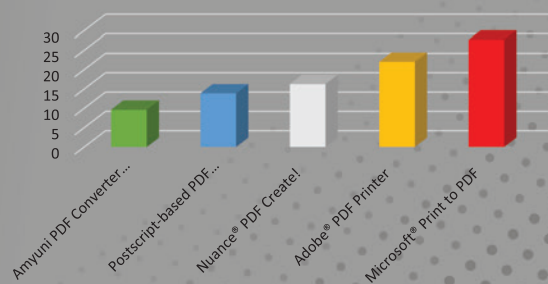
- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment



## High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

Benchmark Testing - Amyuni vs Others  
Seconds required to convert a document to PDF



## Other Developer Components from Amyuni®

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need



AMYUNI   
Technologies

USA and Canada

Toll Free: 1866 926 9864

Support: 514 868 9227

sales@amyuni.com

Europe

UK: 0800-015-4682

Germany: 0800-183-0923

France: 0800-911-248

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.

All development tools available at

[www.amyuni.com](http://www.amyuni.com)



# Reflections on Code First, Persistence and Domain Modeling

Code First is a piece of functionality you find in Entity Framework (EF) that lets you model database tables using plain .NET classes. Frankly, I think the name Code First is a bit misleading, but the work it does under the hood is crystal clear. Code First lays out the structure of the database being used and provides an all-round object-oriented API to work with stored data.

Introduced with EF4.1, Code First is included up through EF6—just one of the approaches you can take to model your database through C# or Visual Basic classes. Up until EF6, you can also use a Visual Studio designer to infer the schema of the database, save it to an XML file with the EDMX extension and create ad hoc classes for use in code. The Visual Studio designer also lets you create an abstract model that's later used to create a physical database.

To make a long story short, up until EF6 there have been two ways of doing the same stuff, but the EDMX approach—although functional—is more problematic than the other. For this reason, the upcoming EF7 discontinues the EDMX support.

Over the years, Code First has been associated with Domain-Driven Design (DDD) and this might have contributed to the general idea that Code First and EDMX are not exactly two ways of doing the same thing. In this column, I'll offer a more architectural perspective of Code First and draw a line between the realm of the domain model and the realm of the persistence model. Code First, along with LINQ, realizes an old dream of most developers: It hides the intricacies of data access (tables, indexes, constraints) behind an object-oriented facade and qualifies as that object-oriented data definition language you never had.

## Historical Background

While working with relational databases, you play by the rules of the SQL language. While coding applications, you play by the rules of the programming language of choice instead. Hence, an abstraction layer is required to bridge the object-oriented—or procedural—nature of top-level programming languages with the SQL language. In the Microsoft .NET Framework, this abstraction layer is the ADO.NET framework.

ADO.NET is a relatively thin abstraction layer in the sense that it only provides objects for your .NET code to place SQL commands. ADO.NET doesn't map any data sent or retrieved from the database to ad hoc object-oriented data structures. In ADO.NET, the tools to get to the data are fully merged with the surrounding .NET Framework, but the data is flat.

About a decade ago, Object/Relational Mapper (O/RM) frameworks appeared on the horizon. An O/RM framework maps the properties of a class to the columns of a table. In doing so, it implements a bunch of design patterns such as Data Mapper, Unit of Work and Query Object. An O/RM framework also maintains internally a set of mapping rules and information about the schema of the target database. This is concrete and tangible information that must be saved somewhere and somehow. NHibernate—the first ever O/RM in the .NET space—stores that information as an XML file. EF initially took the same approach with EDMX files and added a nice designer to manage it from within Visual Studio. Code First maps class properties to columns and tables via either attributes or a fluent (and richer) API.

In a blog post that appeared several months ago, the EF team explained in a clear manner the motivation behind making Code First

the only supported way to store data models in EF7. (You can read the full story at [bit.ly/1sLM3Ur](http://bit.ly/1sLM3Ur).) In the post, the expression “code-based modeling” is used as a more explanatory name for what Code First really does. I couldn't agree more.

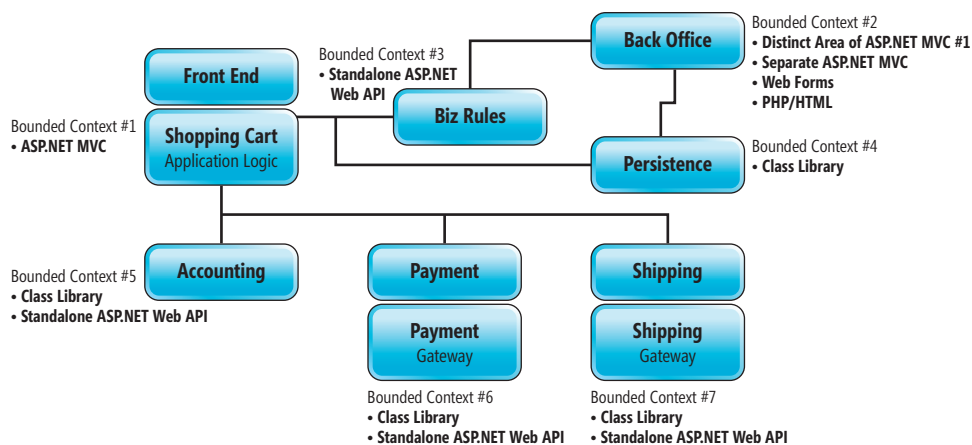


Figure 1 Sample Top-Level Architecture with Bounded Contexts

## DDD in a Nutshell

DDD is an approach to software development that was initially devised as a set of rules applied systematically to control a monumental level of complexity (that is, a huge number of business rules and entities). While DDD shines in



# DevExpress Spreadsheet for WPF & WinForms

## with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.



Free 30-day trial  
[devexpress.com/spreadsheet](http://devexpress.com/spreadsheet)

# #UseTheBest

All trademarks or registered trademarks are property of their respective owners.

very large systems with at least hundreds of rules and entities, it has a lot of value for developers and architects in simpler scenarios. In a nutshell, there's no reason for not applying certain parts of DDD in just about every software project. The part of DDD that's valuable in any project is Strategic Design and is centered on the application of a few well-known methods: Ubiquitous Language, Bounded Context and Context Map. These analytical patterns have little to do

with the actual classes and database tables you end up using in the final application, even though the ultimate goal of using them is to write code more effectively. The DDD strategic design patterns aim at analyzing the business domain and envisioning the top-level architecture of the resulting system. **Figure 1** provides a possible top-level architecture for an e-commerce solution. Each block represents a bounded context identified during analysis and introduced to speed up development.

Each bounded context that comes up from your analysis has its own business language, its own software architecture (including technologies) and its own set of relationships to other bounded contexts. Each bounded context may then be implemented using the software architecture that best fits a given number and skills of the teams involved, budget and time constraints, plus any other stakeholders' concerns such as those related to existing software licenses, costs, expertise, policies, and so on. DDD also provides a clear suggestion for what could be a really effective way to build stuff for a bounded context: the layered architecture.

## The Domain Model in a Layered Architecture

**Figure 2** provides the gist of a layered architecture. It has four layers—ranging from presentation to infrastructure—with an application layer and a domain layer in the middle. In short, it's a generalized form of the well-known three-tier architecture—presentation, business, data—with a neat separation between use-cases logic that changes with the use cases you consider in the presentation and domain logic that's inherent to the specific way of doing business, and is common to all use cases and presentation layers.

The infrastructure layer includes whatever's required to implement and support use cases and persist the state of the domain entities. The infrastructure layer, therefore, includes components that know the connection string to connect to the database.

Central to the DDD approach is the notion of a “domain model.” Quite simply, a domain model is a software model you create to fully represent the business domain. Put another way, it's anything you can do with software that lets you deal with the domain you're facing. Typically, a domain model is populated with entities, events and value objects, and some of the entities and value objects work together to form an indissoluble unit. DDD calls this an “aggregate” and the root of the aggregate is the aggregate root. Persistence occurs at the level of aggregate roots and the aggregate root typically is responsible for persisting all the other entities and value objects in the aggregate.

How would you code an aggregate of entities and value types? It depends on the programming paradigm you're using. Most of the time, a domain model is an object-oriented model where entities

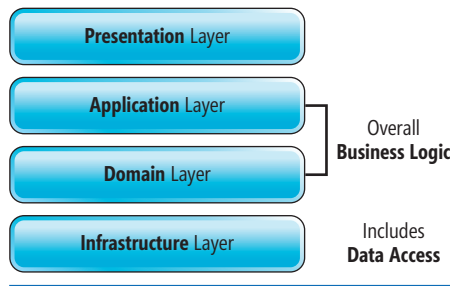


Figure 2 Schema of a Layered Architecture

are classes with properties and methods and value objects are immutable data structures. Using a functional language and immutable data structures is an option, however, at least in certain types of business domains.

Code First is a concrete technology strictly related to the performance of data access tasks. The most characterizing aspect of Code First is the use of classes to represent the underlying schema of tables and the data used by the application. Is the data

used by the application the same as the data persisted by the application through relational tables? Or, asked another way, is the set of classes Code First uses to map tables in the relational database the same as the application's domain model? Well, I'd mostly say no, but when it comes to software architecture, as always, it depends.

## The Domain Model in a Layered Architecture

Code First is sometimes associated with DDD because of its ability to model application's data through classes. While sometimes it's more than acceptable to have a single set of classes that deal with both the business logic of the domain and persistence concerns, in general terms domain model and persistence model are distinct. The domain model is the software model you use to express the domain logic of the system and implement its business rules. It might be an object-oriented model, as well as a functional model or even a plain collection of static methods exposed out of helper classes.

The point of DDD is that you keep persistence concerns off the domain model and in the design of the domain model you focus more on what a business entity does (and how it's used) than on the data it contains and manages. A behavior-centric approach breaks a monumental level of complexity down to a level that can be effectively tackled with code. Let's consider a simple example, a sports match, as shown in **Figure 3**.

To express the behavior of a match entity in the context of a scoring system, you'd model actions like Start, Finish, Goal, Timeout and whatever else makes sense in the specific scenario. These methods implement all business rules and ensure that only actions consistent with the current state of the entity are carried out programmatically. For example, the method Goal would throw if invoked on a Match instance currently suspended because of a timeout. The internal state of the Match entity contains all those properties you'd typically associate

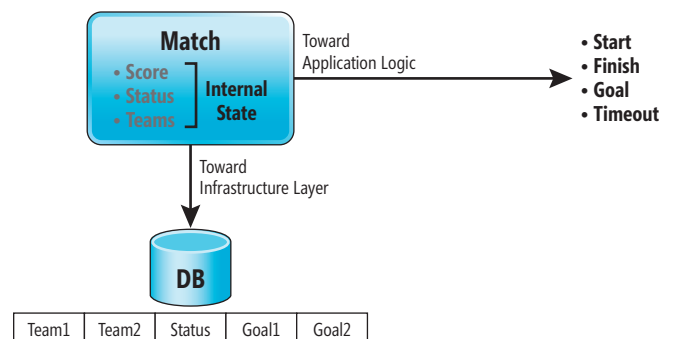


Figure 3 Behavior vs. Data in the Entity of a Domain Model

with such an entity in a pure relational model except that these properties are read-only and updated only internally via methods.

Not all the classes you may have in a domain model must be persisted and persistence might include all properties or just a few. So Code First isn't about domain modeling in general, but its API that maps properties to table columns can be used to persist the classes in your domain model that need be persisted. In this way, you have a single model for the domain that covers both business and persistence needs.

### The Issue of Private Setters

In the domain modeling perspective, you only work with entity-following business workflows as outlined by domain experts. Looking back at the match score example, it might not be consistent with business rules setting the state of the match or the score programmatically. State and score, in fact, change as the workflow makes progress. Likewise, you're not going to have a default parameterless constructor because it would return a Match entity devoid of some critical information such as the names of the playing teams and an ID that would reasonably tie the match to a competition. Yet, if you're using a single model for business and persistence, a parameterless constructor is required; otherwise, EF wouldn't be able to return an instance of the type after a query.

But there's more to consider. When EF performs a query and returns an instance of the Match class, it needs to access the setters of all properties in order to save in the returned instance a state coherent with the information in the database. This legitimate requirement of EF conflicts with the design rules of a domain model. More in general, a way to force a state into an entity of a domain model must exist and most of the time it must be internal and not publicly available via code outside the layer. This is one of the purposes of domain services that, along with the domain, form the Domain Layer of **Figure 2**. If you use Code First, you can achieve the same by simply marking setters as non-public (internal, protected or even private) and adding a default constructor with the same non-public visibility. EF will still find a way (via reflection) to access private members and force a state, but public clients of the domain API won't. Well, not until they use reflection themselves.

### Wrapping Up

It's not unusual to surf the Web and find articles that put Code First in relationship with DDD. Code First is all about persistence of an object-oriented model that's explicitly mapped onto a set of tables. Conceptually speaking, the domain model is completely another thing and even living in a different layer. However, because of some specific capabilities of the Code First API—dealing with private setters and constructors—in some cases it's possible to use a single object-oriented model that includes behavior and business rules and can be easily persisted to a relational database. ■

**DINO ESPOSITO** is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET and Android platforms at JetBrains, and frequent speaker at industry events worldwide, Esposito shares his vision of software at [software2cents@wordpress.com](mailto:software2cents@wordpress.com) and on Twitter: @despos.

**THANKS** to the following Microsoft technical expert for reviewing this article:  
Jon Arne Saeteras

[msdnmagazine.com](http://msdnmagazine.com)



# dtSearch®

## Instantly Search Terabytes of Text

dtSearch's document filters support popular file types, emails with multilevel attachments, databases, web data

**Highlights hits** in all data types; 25+ search options

With APIs for .NET, Java and C++. SDKs for multiple platforms. (See site for articles on faceted search, SQL, MS Azure, etc.)

Visit [dtSearch.com](http://dtSearch.com) for

- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval® since 1991

**dtSearch.com 1-800-IT-FINDS**





# The New Azure DocumentDB Node.js SDK

Over the past year I've been developing a sample app that uses Aurelia on the front end, a server-side API written in Node.js and Azure DocumentDB for its data store. In addition to using Node.js for the server-side API, my app also leverages the Node.js SDK for Azure DocumentDB. Rather than describe the full app, I'll point you to the earlier articles from November ([msdn.com/magazine/mt620011](http://msdn.com/magazine/mt620011)) and December 2015 ([msdn.com/magazine/mt595750](http://msdn.com/magazine/mt595750)) when I wrote about this application. You can even download the original application and compare it to the new source that reflects the changes described in this article. And because I'm frequently tweaking the app, you can always take a look at the GitHub repository at [bit.ly/25crZAG](http://bit.ly/25crZAG).

Because of changes to Aurelia, to many of the Node.js packages I'm using, to DocumentDB features and even to the aforementioned SDK over the past months, it was time to do a number of updates, and not just to packages, but also update my code to take advantage of newer features throughout. I won't address updates to Aurelia here; instead, I'll keep my focus on changes in DocumentDB and on modifying my Node.js API code to benefit from those changes.

## Step 1: Implementing Promises for Async Calls

The first thing I did was run "npm update" on my Node.js project. The update went well, but running the app afterward was less successful. I quickly encountered an error that told me my use of callbacks had become a problem. Somewhere in the depths of dependencies, an API now favors JavaScript promises over callbacks. The paradigm of promises (akin to `async/await` in .NET) has been around for a while but I had taken the familiar path of using callbacks when creating the sample. Now it was time to dig in my heels, hold my breath and replace all of the callbacks in the Node.js API with promises. Unfortunately, this wasn't just a matter of replacing terms, but it required changing the actual structure of the code. In the layers of my API, I was using callbacks in the `DocDBUtils` file that talked directly to the Node.js SDK for DocumentDB. And I was using callbacks in the `DocDBDao` class that talked to the utilities (in `DocDBUtils`) and to the SDK. This meant that when communicating with the utilities I had a layered system of callbacks. Finally, the `ninja.js` class made calls into the `DocDBDao` class to trigger data retrieval or updates. These methods also used callbacks and depended on the callbacks of the lower files. So I needed to implement promises from the bottom (`DocDBUtils`) up.

There are a number of JavaScript APIs that help with the implementation of promises. One is called `Q` and the DocumentDB team created a wrapper for its Node.js SDK that uses `Q` and, therefore, makes using promises when coding against DocumentDB a much easier task.

This wrapper, `documentdb-q-promises`, is on GitHub at [bit.ly/1pWHHCE](http://bit.ly/1pWHHCE).

Therefore, my first step was to install `Q` using the node package manager (npm):

```
npm install documentdb-q-promises
```

Then, in all of the node classes that were using the base SDK (the aforementioned classes, as well as one called `api.js`) I had to modify the "require" statements that were originally letting my classes know to use the `DocumentClient` class from the initial SDK:

```
var DocumentDBClient = require('documentdb').DocumentClient;
```

to point to the `DocumentClientWrapper` class from the new API:

```
var DocumentDBClient = require('documentdb-q-promises').DocumentClientWrapper;
```

The `DocDBUtils` class requires an additional reference to the `Q` library directly, so its `DocumentDBClient` is defined as:

```
var DocumentClient = require('documentdb-q-promises').DocumentClientWrapper,
    Q = require("q");
```

Next, I had to refactor the callback code to use the promises. I struggled with this for a while until I had the pattern down. Then, with some working functions in the `DocDBUtils` class, I was able to more easily fix up the functions in the classes that call into this class. But before I got to this point, it was definitely an arduous process: change code, debug, read the errors, scratch my head, Google some more and change the code again. There was a bit of griping on Twitter, as well, so my friends kept me from hurting my head too much. This wasn't so much because it's terribly difficult, but just because—regardless of my programming experience—I'm still something of a noob in JavaScript.

As an example, I'll begin with the very first function to be hit when the API is run: the `init` method in `DocDBDao.js`. This method makes sure that the rest of the API is aware of the DocumentDB account, connects to it using the authentication keys and knows the name of the database, as shown in **Figure 1**.

**Figure 1 The Original `getOrCreateDatabase` Method Using Callbacks**

```
getOrCreateDatabase: function (client, databaseId, callback) {

    var querySpec = { *query for database name defined here* };
    client.queryDatabases(querySpec).toArray(function (err, results) {
        if (err) {
            callback(err);
        } else {
            if (results.length === 0) {
                var databaseSpec = {
                    id: databaseId
                };
                client.createDatabase(databaseSpec, function (err, created) {
                    callback(null, created);
                });
            } else {
                callback(null, results[0]);
            }
        }
    });
},
```

Code download available at [msdn.com/magazine/0716magcode](http://msdn.com/magazine/0716magcode).

getOrCreateDatabase is called from the init function in the docDb-Dao class. The parameter named client is an instance of DocumentDB-Client from the original SDK. The third parameter, named callback, refers to the calling function—in this case, init. The getOrCreateDatabase method defines a query in the querySpec variable and then calls client.queryDatabase with the query. If queryDatabase returns an error, getOrCreateDatabase passes that error back up to the calling function via the callback. Otherwise it inspects the results. If results.length is 0, it creates a new database and then passes information returned by createDatabase back to the calling function. If results.length isn't 0, the first item in the results array is returned back in the callback.

Now let's have a look at this same function, shown in **Figure 2**, rewritten to use promises (remember, these are like async/await in the Microsoft .NET Framework), leveraging the promises implementation provided by Q.

The first thing to notice is that there's no callback in the parameter list for the function. After defining the query, the function makes the call to queryDatabases, but not like before. This time, I'm using the queryDatabases wrapper defined by the new SDK. Rather than calling toArray on queryDatabases, I used the toArrayAsync method, which is one of a number of asynchronous methods provided by the documentdb-q-promises SDK. toArrayAsync returns an instance of a *promise* type defined by the Q library; promise has a "then" method (similar to the await you might be familiar with from the .NET Framework) that allows you to define a function to execute when the queryDatabases.toArrayAsync call completes. The first bit of logic indicates what to do if the call is successful. Just like before, I check to see if the length is 0, indicating that the database doesn't yet exist. If this is the case, then I create a new database, but this time using the createDatabaseAsync method, which, like the other async methods, returns a promise object. If the database is created successfully, I then process the database response. I've left out some of the additional logic around creating the database, but you can see it if you download the example code.

The next part of the method specifies what should happen if the query does find a database, which is simply to return the first item in the results. The results of toArrayAsync contain a feed property that wraps the results, which is why you see the syntax as results.feed[0].

Last, if the queryDatabases call fails, the function returns an error.

Now that you've walked through this, let's look at the pattern again:

```
CallToAsyncFunction().then(function to execute when complete){
    success logic
},
function(err) {failure logic}
);
```

You call one of the asynchronous methods and use its then method to define a nameless function to execute when the call completes. In the function's logic, you first specify code to execute when the method has succeeded (optionally returning some result) and then code to execute if the method fails (also with the option of returning a result).

I've implemented this pattern throughout the revised API, replacing all of the callback constructs in the three models.

However, I can't simply debug now and expect this to work because I have a waterfall of promises starting with the ninjas class, which doesn't yet know about the new documentdb-q-promises SDK. You can try to replace those other callbacks in the original sample yourself, or see the fully updated solution in the download.

Now my Node.js interaction with DocumentDB is using recommended technology, so let's look at some other features in DocumentDB and I'll discuss how I implemented them in the API.

## Parameterized Queries

In the first iteration of my sample, I did something in the ninja.js class that I'd never do in my .NET apps—I hacked query strings together with string concatenation. At least I was using the ES6-enabled string interpolation to do that concatenation, but still I'm a little ashamed and have no excuse. Except perhaps two. The first is I was learning from the provided samples and not yet using my brain. (Does this even count?) The second is that security wasn't paramount at the moment because performing a SQL attack on an Azure DocumentDB isn't that much of an issue due to the way queries work in the database. Even the documentation says that DocumentDB isn't really susceptible to the most common types of injection attacks, though there's always a chance of an evildoer finding a way to take advantage of injection. Still, it's always good to be extra cautious about security, and parameterized queries have been a recommended practice for data access for a very long time.

In the earlier version of the sample, a filter function defined a type called querySpec with a property named query. The query property value was SQL, used to retrieve a set of ninjas from DocumentDB:

```
var querySpec = {
  query:
    'SELECT ninja.id, ninja.Name,ninja.ServedInOniwaban,ninja.DateOfBirth
  FROM ninja'
```

The filter function reads a filter value contained in the URL. For example, when the user searches for every ninja whose name contains "San," the URL is localhost:9000/api/ninjas?q=San. The original function constructed a query predicate by simply concatenating the filter value, found in request.query.q, to the predicate:

```
q = ' WHERE CONTAINS(ninja.Name,'" + request.query.q + "'");
```

I then appended the predicate to the base query, which was stored in querySpec.query.

Even though injection attacks aren't nearly as easy using a filter value, I've replaced that bit of logic with a parameter DocumentDB will comprehend. Rather than concatenate the filter value entered by the end user (San), I'll use a parameter placeholder called @namepart in the predicate. Then I'll add a new property to querySpec called parameters and, with JSON formatting, define it using name and

**Figure 2 getOrCreateDatabase Using Promises**

```
getOrCreateDatabase: function (client, databaseId) {

  var querySpec = { *query for database name defined here* };

  return client.queryDatabases(querySpec).toArrayAsync().then(function (results) {
    if (results.length === 0) {
      var databaseSpec = {
        id: databaseId
      };
      client.createDatabaseAsync(databaseSpec)
        .then(function (databaseResponse) {
          db = databaseResponse.resource;
          return client.createCollectionAsync(db._self, collectionDefinition);
        })
    }
    return results.feed[0];
  },
  function (err) { return err; }
  );
}
```

value properties that DocumentDB will look for. I can then specify the parameter name and the query value passed in by the URL:

```
querySpec.query += " WHERE CONTAINS(ninja.Name, @namepart)";
querySpec.parameters = [{
  name: '@namepart',
  value: request.query.q
}]
```

DocumentDB will then execute this as a parameterized query so any evil SQL will be unable to hurt my data.

## Goodbye Self-Links and Don't Let the Door Hit You on the Way Out

OK, so that's a bit harsh, but it's how many of us felt about the need to use selfLinks from every type of object, whether it was a database, a collection, a document or other object in DocumentDB. The selfLink value is how the object identified itself in DocumentDB. You'd have to query DocumentDB with a known identifier—database or collection name or the identity value of a document—in order to get its selfLink so you could perform other operations. SelfLinks are still there, but you no longer need them to interact with an object. If you know the details to build up a link to an object, you can use that instead of the selfLink. I'll demonstrate that shortly in combination with the next feature I've taken advantage of in my revised sample: Upserts.

## Replacing Replace with Upsert

I was eager to remove the clunky update function in my API, which required that I first retrieve the item to be updated from the database in order to:

1. Ensure that it existed
2. Get access to its selfLink
3. Get access to the full document in case the item passed into the update method has a limited schema

Then I had to update fields from the document retrieved from the database with values from the item passed to the update method from the client application. Finally, I had to tell DocumentDB to replace the existing document in the database with this modified document. You can visit the earlier article or sample to take a look at the update-Item function in docDbDao if you're curious what this all looked like.

Luckily, in October Microsoft announced the addition of atomic upsert functionality to DocumentDB, which enables it to figure out if a given document needs to be inserted or updated. See the related blog post at [bit.ly/1G5wtpY](http://bit.ly/1G5wtpY) for a more detailed explanation.

Upsert allows me to do a simple update using the document I have at hand. The documentdb-q-promises SDK provides an asynchronous version of replace. Here's my revised update function:

```
updateItem: function (item) {
  var self = this;
  var docLink = "dbs/Ninjas/colls/Ninjas";
  item.DateModified = Date.now();
  return self.client.upsertDocumentAsync(docLink, item).then(function (replaced) {
    return replaced;
  },
  function (err) {
    return err;
  });
},
```

Notice the docLink value that I'm building. This is the new feature I mentioned that helps me avoid needing the actual SelfLink from the database for the document I want to update. I'm simply specifying that

the database is named Ninjas and the collection also happens to be named Ninjas. I pass the docLink value along with the item that came from the client to the upsertDocumentAsync command, and then pass back the Boolean (replaced) that's returned when the command has been executed. Notice also that along with the async command, I've modified this logic to leverage the promise returned by the Async method. You can tell because I'm composing *then* on the Async method.

## So Much More Has Been Added to DocumentDB

While my little sample takes advantage of only a few of its new capabilities, so much more has been added to DocumentDB since I wrote my earlier columns. More SQL commands are in there, including TOP for paging and ORDER BY for sorting. ORDER BY is dependent on indexes on collections, which makes sense because this is about Big Data here, and you do need to tune the database to meet your particular needs. It's also possible to modify the indexing policies in existing collections rather than having to live with an unfortunate previous choice. If you're using the .NET Client API for DocumentDB, the LINQ provider has become much richer, which you can read about at [bit.ly/10d3ia2](http://bit.ly/10d3ia2).

DocumentDB is for persisting large amounts of data, and Microsoft is working to make access more efficient and cost-effective. To that end, the company introduced partition keys, a core feature of DocumentDB that allows you to partition collections to accommodate a "large volume of data at high rates or applications that require high throughput, low latency access to data." You can read more about partition keys at [bit.ly/1Tp0xFj](http://bit.ly/1Tp0xFj).

Microsoft also addressed pricing issues that kept users from creating additional collections because fees were related to the number of collections. New plans are based on the volume of data and throughput, so you can use more collections and not worry quite as much about excessive costs for collections that don't have a lot of activity. But each collection still has a minimum throughput. The team continues to look for ways to improve resource usage so they can lower these minimums in the future. For more information about the new pricing plans, visit [bit.ly/1jgnbn9](http://bit.ly/1jgnbn9).

DocumentDB tools have also changed, and they'll continue to do so in the future. There are more ways to explore your DocumentDB in the Azure Management Portal, and the data migration tools have acquired more capabilities. One change I was especially happy to see was that the Cloud Explorer extension for Visual Studio now supports connecting to and exploring DocumentDB data. You can even edit and save changes to the raw data through the Cloud Explorer, though at this time, querying is not an option.

To keep up with the growing feature set, keep an eye on the DocumentDB-tagged blog posts on the Azure blog ([bit.ly/1Y5T1SB](http://bit.ly/1Y5T1SB)) and follow the @documentdb Twitter account for updates. ■

---

**JULIE LERMAN** is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other .NET topics at user groups and conferences around the world. She blogs at [thedatafarm.com/blog](http://thedatafarm.com/blog) and is the author of "Programming Entity Framework," as well as a Code First and a DbContext edition, all from O'Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at [juliel.me/PS-Videos](http://juliel.me/PS-Videos).

---

**THANKS** to the following Microsoft technical expert for reviewing this article:

Andrew Liu



# We didn't invent the Internet...

...but our components help you power the apps that bring it to business.



## TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**  
AS2, EDI/X12, NAESB, OFTP ...
- **Credit Card Processing**  
Authorize.Net, TSYS, FDMS ...
- **Shipping & Tracking**  
FedEx, UPS, USPS ...
- **Accounting & Banking**  
QuickBooks, OFX ...
- **Internet Business**  
Amazon, eBay, PayPal ...
- **Internet Protocols**  
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**  
SSH, SFTP, SSL, Certificates ...
- **Secure Email**  
S/MIME, OpenPGP ...
- **Network Management**  
SNMP, MIB, LDAP, Monitoring ...
- **Compression & Encryption**  
Zip, Gzip, Jar, AES ...



## The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity  
powered by 

To learn more please visit our website →

[www.nsoftware.com](http://www.nsoftware.com)

# Better Data Binding in .NET

Mark Sowul

**Data binding** is a powerful technique for developing UIs: It makes it easier to separate view logic from business logic, and easier to test the resulting code. Although present in the Microsoft .NET Framework since the beginning, data binding became more prominent with the advent of Windows Presentation Foundation (WPF) and XAML, as it forms the “glue” between the View and ViewModel in the Model-View-ViewModel (MVVM) pattern.

The drawback of implementing data binding has always been the need for magic strings and boilerplate code, both to broadcast changes to the properties and to bind UI elements to them. Over the years, various toolkits and techniques have come along to lessen the pain; this article aims to simplify the process even further.

First, I'll review the basics of implementing data binding, as well as common techniques to simplify it (if you're already familiar with

the subject, feel free to skip those sections). After that, I'll develop a technique you might not have considered (“A Third Way”), and introduce solutions to related design difficulties encountered when developing applications using MVVM. You can obtain the finished version of the framework I develop here in the accompanying code download, or add the SolSoft.DataBinding NuGet package to your own projects.

## The Basics: INotifyPropertyChanged

Implementing `INotifyPropertyChanged` is the preferred way to enable an object to be bound to a UI. It's simple enough, containing just one member: the `PropertyChanged` event. The object should raise this event when a bindable property changes, in order to notify the view that it should refresh its representation of the property's value.

The interface is simple, but implementing it isn't. Manually raising the event with hardcoded textual property names isn't a solution that scales well, nor does it stand up to refactoring: You must take care to ensure the textual name remains in sync with the property name in the code. This will not endear you to your successors. Here's an example:

```
public int UnreadItemCount
{
    get
    {
        return m_unreadItemCount;
    }
    set
    {
        m_unreadItemCount = value;
        OnNotifyPropertyChanged(
            new PropertyChangedEventArgs("UnreadItemCount")); // Yuck
    }
}
```

### This article discusses:

- Implementing `INotifyPropertyChanged`
- Getting the name of code elements
- Refactoring support
- Object-oriented design principles
- Data binding with interrelated properties

### Technologies discussed:

Microsoft .NET Framework, Data Binding, XAML

### Code download available at:

[msdn.com/magazine/0716magcode](http://msdn.com/magazine/0716magcode)

There are several techniques people have developed in response, in order to maintain their sanity (see, for example, the Stack Overflow question at [bit.ly/24ZQ7CY](http://bit.ly/24ZQ7CY)); most of them fall into one of two types.

## Common Technique 1: Base Class

One way to simplify the situation is with a base class, in order to reuse some of the boilerplate logic. This also provides a few ways to obtain the property name programmatically, instead of having to hard code it.

**Getting the Property Name with Expressions:** The .NET Framework 3.5 introduced expressions, which allow for runtime inspection of code structure. LINQ uses this API to great effect, for example, to translate .NET LINQ queries into SQL statements. Enterprising developers have also leveraged this API to inspect property names. Using a base class to do this inspection, the preceding setter could be rewritten as:

```
public int UnreadItemCount
{
    ...
    set
    {
        m_unreadItemCount = value;
        RaiseNotifyPropertyChanged(() => UnreadItemCount);
    }
}
```

In this way, renaming `UnreadItemCount` will also rename the expression reference, so the code will still work. The signature of `RaiseNotifyPropertyChanged` would be as follows:

```
void RaiseNotifyPropertyChanged<T>(Expression<Func<T>> memberExpression)
```

Various techniques exist for retrieving the property name from the `memberExpression`. The C# MSDN blog at [bit.ly/25baMHM](http://bit.ly/25baMHM) provides a simple example:

```
public static string GetName<T>(Expression<Func<T>> e)
{
    var member = (MemberExpression)e.Body;
    return member.Member.Name;
}
```

StackOverflow presents a more thorough listing at [bit.ly/23Xczu2](http://bit.ly/23Xczu2). In any case, there's a downside to this technique: Retrieving the name of the expression uses reflection, and reflection is slow. The performance overhead can be significant, depending on how many property change notifications there are.

**Getting the Property Name with CallerMemberName:** C# 5.0 and the .NET Framework 4.5 brought an additional way to retrieve the property name, using the `CallerMemberName` attribute (you can use this with older versions of the .NET Framework via the `Microsoft.Bcl.NuGet` package). This time, the compiler does all the work, so there's no runtime overhead. With this approach, the method becomes:

```
void RaiseNotifyPropertyChanged<T>([CallerMemberName] string propertyName = "")
```

And the call to it is:

```
public int UnreadItemCount
{
    ...
    set
    {
        m_unreadItemCount = value;
        RaiseNotifyPropertyChanged();
    }
}
```

The attribute instructs the compiler to fill in the caller name, `UnreadItemCount`, as the value of the optional parameter `propertyName`.

**Getting the Property Name with nameof:** The `CallerMemberName` attribute was probably tailor-made for this use case (raising `PropertyChanged` in a base class), but in C# 6, the compiler team

finally provided something much more broadly useful: the `nameof` keyword. `nameof` is handy for many purposes; in this case, if I replace the expressions-based code with `nameof`, once again the compiler will do all the work (no runtime overhead). It's worth noting this is strictly a compiler version feature and not a .NET version feature: You could use this technique and still target the .NET Framework 2.0. However, you (and all your team members) have to be using at least Visual Studio 2015. Using `nameof` looks like this:

```
public int UnreadItemCount
{
    ...
    set
    {
        m_unreadItemCount = value;
        RaiseNotifyPropertyChanged(nameof(UnreadItemCount));
    }
}
```

There's a general problem, though, with any base class technique: It "burns your base class," as the saying goes. If you want your view model to extend a different class, you're out of luck. It also does nothing to handle "dependent" properties (for example, a `FullName` property that concatenates `FirstName` and `LastName`: Any change to `FirstName` or `LastName` must also trigger a change on `FullName`).

## Common Technique 2: Aspect-Oriented Programming

Aspect-oriented programming (AOP) is a technique that basically "post-processes" your compiled code, either at run time or with a post-compilation step, in order to add certain behaviors (known as an "aspect"). Usually, the aim is to replace repetitive boilerplate code, such as logging or exception handling (so-called "cross-cutting concerns"). Not surprisingly, implementing `INotifyPropertyChanged` is a good candidate.

There are several toolkits available for this approach. `PostSharp` is one ([bit.ly/1Xmq4n2](http://bit.ly/1Xmq4n2)). I was pleasantly surprised to learn that it properly handles dependent properties (for example, the `FullName` property described earlier). An open source framework called "Fody" is similar ([bit.ly/1wXR2VA](http://bit.ly/1wXR2VA)).

Manually raising the event  
with hardcoded textual  
property names will not endear  
you to your successors.

This is an attractive approach; its drawbacks might not be significant. Some implementations intercept behavior at run time, which incurs a performance cost. The post-compilation frameworks, in contrast, shouldn't introduce any runtime overhead, but might require some sort of install or configuration. `PostSharp` is currently offered as an extension to Visual Studio. Its free "Express" edition limits use of the `INotifyPropertyChanged` aspect to 10 classes, so this likely means a monetary cost. `Fody`, on the other hand, is a free NuGet package, which makes it appear to be a compelling choice. Regardless, consider that with any AOP framework the code you write isn't exactly the same code you'll be running ... and debugging.

## A Third Way

An alternative way of handling this is to leverage object-oriented design: Have the properties themselves be responsible for raising the events! It's not a particularly revolutionary idea, but it's not one I've encountered outside of my own projects. In its most basic form, it might look like the following:

```
public class NotifyProperty<T>
{
    public NotifyProperty(INotifyPropertyChanged owner, string name, T initialValue);

    public string Name { get; }
    public T Value { get; }
    public void SetValue(T newValue);
}
```

The idea is that you provide the property with its name and a reference to its owner, and let it do the work of raising the PropertyChanged event—something like:

```
public void SetValue(T newValue)
{
    if(newValue != m_value)
    {
        m_value = newValue;
        m_owner.PropertyChanged(m_owner, new PropertyChangedEventArgs(Name));
    }
}
```

The problem is that this won't actually work: I can't raise an event from another class like that. I need some sort of contract with the owning class to allow me to raise its PropertyChanged event: that's exactly the job of an interface, so I'll create one:

```
public interface IRaisePropertyChanged
{
    void RaisePropertyChanged(string propertyName)
}
```

Once I have this interface, I can actually implement NotifyProperty.SetValue:

```
public void SetValue(T newValue)
{
    if(newValue != m_value)
    {
        m_value = newValue;
        m_owner.RaisePropertyChanged(this.Name);
    }
}
```

**Implementing IRaisePropertyChanged:** Requiring the property owner to implement an interface does mean that each view model

### Figure 1 Code Required to Implement IRaisePropertyChanged

```
// PART 1: required for any class that implements INotifyPropertyChanged
public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(PropertyChangedEventArgs args)
{
    // In C# 6, you can use PropertyChanged?.Invoke.
    // Otherwise I'd suggest an extension method.
    var toRaise = PropertyChanged;
    if (toRaise != null)
        toRaise(this, args);
}

// PART 2: IRaisePropertyChanged-specific
protected virtual void RaisePropertyChanged(string propertyName)
{
    OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
}

// This method is only really for the sake of the interface,
// not for general usage, so I implement it explicitly.
void IRaisePropertyChanged.RaisePropertyChanged(string propertyName)
{
    this.RaisePropertyChanged(propertyName);
}
```

class will require some boilerplate, as illustrated in **Figure 1**. The first part is required for any class to implement INotifyPropertyChanged; the second part is specific to the new IRaisePropertyChanged. Note that because the RaisePropertyChanged method isn't intended for general use, I prefer to implement it explicitly.

I could put this boilerplate in a base class and extend it, which seems to bring me back to my earlier discussion. After all, if I apply CallerMemberName to the RaisePropertyChanged method, I've basically reinvented the first technique, so what's the point? In both cases, I could just copy the boilerplate to other classes if they can't derive from a base class.

There's a general problem, though, with any base class technique: It "burns your base class," as the saying goes.

One key difference compared to the earlier base class technique is that in this case there's no real logic in the boilerplate; all the logic is encapsulated in the NotifyProperty class. Checking whether the property value has changed before raising the event is simple logic, but it's still better not to duplicate it. Consider what would happen if you wanted to use a different IEqualityComparer to do the check. With this model, you'd need to alter only the NotifyProperty class. Even if you had multiple classes with the same IRaisePropertyChanged boilerplate, each implementation could benefit from the changes to NotifyProperty without having to change any code itself. Regardless of any behavior changes you might wish to introduce, the IRaisePropertyChanged code is very unlikely to change.

**Putting the Pieces Together:** Now I have the interface the view model needs to implement, and the NotifyProperty class used for the properties that will be data bound. The last step is constructing the NotifyProperty; for that, you still need to pass in a property name, somehow. If you're lucky enough to be using C# 6, this is easily done with the nameof operator. If not, you can instead create the NotifyProperty with the aid of expressions, such as by using an extension method (unfortunately, there's nowhere for CallerMemberName to help this time):

```
public static NotifyProperty<T> CreateNotifyProperty<T>(
    this IRaisePropertyChanged owner,
    Expression
```

With this approach, you'll still pay a reflection cost, but only when creating an object, rather than every time a property changes. If that's still too expensive (you're creating many objects), you can always cache a call to GetName, and keep that as a static readonly value in the view model class. For either case, **Figure 2** shows an example of a simple view model.

# We Made WPF GREAT!



## Xceed Business Suite for WPF

- 104 high-quality WPF controls and themes
- The fastest and most advanced datagrid
- Widely used controls, over 500K downloads
- Created by WPF pioneers
- 10 years of WPF excellence
- Frequently updated and improved





**Binding and Renaming:** While I'm talking about names, it's a good time to discuss another data-binding concern. Safely raising the PropertyChanged event without a hardcoded string is half the battle to survive refactoring; the data binding itself is the other half. If you rename a property that's used for a binding in XAML, success is, shall I say, not guaranteed (see, for example, [bit.ly/1WCWE5m](http://bit.ly/1WCWE5m)).

The alternative is to code the data bindings manually in the code-behind file. For example:

```
// Constructor
public LogInDialog()
{
    InitializeComponent();

    LogInViewModel forNaming = null;
    m_textBoxUserName.SetBinding(TextBox.TextProperty,
        ObjectNamingExtensions.GetName(() => forNaming.UserName);

    // Or with C# 6, just nameof(LogInViewModel.UserName)
}
```

It's a little weird to have that null object solely for leveraging the expressions functionality, but it does work (you don't need it if you have access to nameof).

I find this technique valuable, but I do recognize the tradeoffs. On the plus side, if I rename the UserName property, I can be confident that the refactoring will work. Another significant benefit is that "Find All References" works just as expected.

On the minus side, it's not necessarily as simple and natural as doing the binding in XAML, and it prevents me from keeping the UI design "independent." I can't just redesign the appearance in the Blend tool without changing code, for example. Additionally, this technique doesn't work against data templates; you can extract that template into a custom control, but that's more effort.

Safely raising the  
PropertyChanged event without  
a hardcoded string is half the  
battle to survive refactoring; the  
data binding itself is the other half.

In total, I gain flexibility to change the "data model" side, at the cost of flexibility on the "view" side. Overall, it's up to you whether the advantages justify declaring the bindings this way.

## "Derived" Properties

Earlier, I described a scenario in which it's particularly inconvenient to raise the PropertyChanged event, namely for those properties whose value depends on other properties. I mentioned the simple example of a FullName property that depends on FirstName and LastName. My goal for implementing this scenario is to take in those base NotifyProperty objects (FirstName and LastName), as well as the function to calculate the derived value from them (for example, FirstName.Value + " " + LastName.Value), and with that, produce a property object that will automatically handle the rest

Figure 2 A Basic ViewModel with a NotifyProperty

```
public class LogInViewModel : IRaisePropertyChanged
{
    public LogInViewModel()
    {
        // C# 6
        this.m_usernameProperty = new NotifyProperty<string>(
            this, nameof(Username), null);
        // Extension method using expressions
        this.m_usernameProperty = this.CreateNotifyProperty(() => Username, null);
    }

    private readonly NotifyProperty<string> m_usernameProperty;

    public string Username
    {
        get
        {
            return m_usernameProperty.Value;
        }
        set
        {
            m_usernameProperty.SetValue(value);
        }
    }

    // Plus the IRaisePropertyChanged code in Figure 1 (otherwise, use a base class)
}
```

for me. To enable this, there are a couple of tweaks I'll make to my original NotifyProperty.

The first task is to expose a separate ValueChanged event on NotifyProperty. The derived property will listen to this event on its underlying properties, and respond by calculating a new value (and raising the appropriate PropertyChanged event for itself). The second task is to extract an interface, IProperty<T>, to encapsulate the general NotifyProperty functionality. Among other things, this allows me to have derived properties come from other derived properties. The resulting interface is straightforward and is listed here (the corresponding changes to NotifyProperty are very simple, so I won't list them):

```
public interface IProperty<TValue>
{
    string Name { get; }

    event EventHandler<ValueChangedEventArgs> ValueChanged;

    TValue Value { get; }
}
```

Creating the DerivedNotifyProperty class seems straightforward, too, until you start trying to put the pieces together. The basic idea was to take in the underlying properties and a function to calculate some new value from them, but that immediately runs into trouble because of generics. There's no practical way to take in multiple different property types:

```
// Attempted constructor
public DerivedNotifyProperty(IRaisePropertyChanged owner,
    string propertyName, IProperty<T1> property1, IProperty<T2> property2,
    Func<T1, T2, TDerived> derivedValueFunction)
```

I can get around the first half of the issue (accepting multiple generic types) by using static Create methods instead:

```
static DerivedNotifyProperty<TDerived> CreateDerivedNotifyProperty
<T1, T2, TDerived>(this IRaisePropertyChanged owner,
    string propertyName, IProperty<T1> property1, IProperty<T2> property2,
    Func<T1, T2, TDerived> derivedValueFunction)
```

But the derived property still needs to listen for the ValueChanged event on each base property. Solving this requires two steps. First, I'll extract the ValueChanged event into a separate interface:



Cross-Platform, Native Mobile Controls

Download trial @  
[goxuni.com](http://goxuni.com)

Supports iOS, Android and Xamarin

Built-in animation. Deliver great mobile experiences with animation

Flexible grids, charts, calendars, and gauges

Develop in the IDE of your choice

**xuni**

# Cross-Platform Data Visualization

GrapeCity

High-performance charts and grids

Full Angular 2 support

Zero dependencies

TypeScript source code

**wijmo**

Country ID	Downloads	Expenses
US	17,730	2,078.36
Germany	8,517	7,298.40
Japan	12,416	2,702.37
Italy	1,032	9,264.42
Spain	8,973	8,716.44
France	17,67	7,754.18
UK	15,074	3,012.49
Canada	6,266	1,065.69
Australia	17,495	7,095.09
India	7,000	7,704.11
China	3,865	1,761.09
South Korea	13,282	9,180.40
Japan	29	9,323.68
Italy	148	2,693.02
Greece	10,771	7,362.19
Spain	2,934	1,243.09
US		
Germany		

Next-Gen JavaScript UI Controls

©2016 GrapeCity, Inc. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.

Download trial @  
[wijmo.com](http://wijmo.com)

```
public interface INotifyValueChanged // No generic type!
{
    event EventHandler<ValueChangedEventArgs> ValueChanged;
}

public interface IProperty<TValue> : INotifyValueChanged
{
    string Name { get; }
    TValue Value { get; }
}
```

This allows the `DerivedNotifyProperty` to take in the non-generic `INotifyValueChanged`, instead of the generic `IProperty<T>`. Second, I need to calculate the new value without generics: I'll take the original `derivedValueFunction` that accepts the two generic parameters and from that create a new anonymous function that doesn't require any parameters—instead, it will reference the values of the two properties passed in. In other words, I'll create a closure. You can see this process in the following code:

```
static DerivedNotifyProperty<TDerived> CreateDerivedNotifyProperty
<T1, T2, TDerived>(this IRaisePropertyChanged owner,
    string propertyName, IProperty<T1> property1, IProperty<T2> property2,
    Func<T1, T2, TDerived> derivedValueFunction)
{
    // Closure
    Func<TDerived> newDerivedValueFunction =
        () => derivedValueFunction (property1.Value, property2.Value);

    return new DerivedNotifyProperty<TValue>(owner, propertyName,
        newDerivedValueFunction, property1, property2);
}
```

The new “derived value” function is just `Func<TDerived>` with no parameters; now the `DerivedNotifyProperty` requires no knowledge of the underlying property types, so I can happily create one from multiple properties of different types.

The other subtlety is when to actually call that derived value function. An obvious implementation would be to listen for the `ValueChanged` event of each underlying property and call the function whenever a property changes, but that's inefficient when multiple underlying properties change in the same operation (imagine a “Reset” button that clears out a form). A better idea is to produce the value on demand (and cache it), and invalidate it if any of the underlying properties change. `Lazy<T>` is a perfect way to implement this.

There's nothing that forces  
you to use this technique  
only in the context of a UI  
view model.

You can see an abbreviated listing of the `DerivedNotifyProperty` class in **Figure 3**. Notice that the class takes in an arbitrary number of properties to listen to—although I list only the `Create` method for two underlying properties, I create additional overloads that take in one underlying property, three underlying properties and so on.

Note that the underlying properties could come from different owners. For example, assume you have an `Address` view model with an `IsValid` property. You also have an `Order` view model that contains two `Address` view models, for billing and shipping addresses. It would be reasonable to create an `IsValid` property

on the parent `Order` view model that combines the `IsValid` properties of the child `Address` view models, so you can submit the order only if both addresses are valid. To do this, the `Address` view model would expose both `bool IsValid { get; }` and `IProperty<bool> IsValidProperty { get; }`, so the `Order` view model can create a `DerivedNotifyProperty` that references the child `IsValidProperty` objects.

## The Usefulness of DerivedNotifyProperty

The `FullName` example I gave for a derived property is fairly contrived, but I do want to discuss some real use cases and tie them to some design principles. I just touched on one example: `IsValid`. This is a fairly simple and powerful way to disable the “Save” button on a form, for example. Note that there's nothing that forces you to use this technique only in the context of a UI view model. You can use it to validate business objects, too; they just need to implement `IRaisePropertyChanged`.

A second situation where derived properties are extremely useful is in “drill down” scenarios. As a simple example, consider a combo box for selecting a country, where selecting a country populates a list of cities. You can have `SelectedCountry` be a `NotifyProperty` and, given a `GetCitiesForCountry` method, create `AvailableCities` as a `DerivedNotifyProperty` that will automatically stay in sync when the selected country changes.

A third area where I've used `NotifyProperty` objects is for indicating whether an object is “busy.” While the object is considered busy, certain UI features should be disabled, and perhaps the user

Figure 3 Core Implementation of `DerivedNotifyProperty`

```
public class DerivedNotifyProperty<TValue> : IProperty<TValue>
{
    private readonly IRaisePropertyChanged m_owner;
    private readonly Func<TValue> m_getValueProperty;

    public DerivedNotifyProperty(IRaisePropertyChanged owner,
        string derivedPropertyName, Func<TValue> getDerivedPropertyValue,
        params INotifyValueChanged[] valueChangesToListenFor)
    {
        this.m_owner = owner;
        this.Name = derivedPropertyName;

        this.m_getValueProperty = getDerivedPropertyValue;
        this.m_value = new Lazy<TValue>(m_getValueProperty);

        foreach (INotifyValueChanged valueChangeToListenFor in valueChangesToListenFor)
            valueChangeToListenFor.ValueChanged += (sender, e) => RefreshProperty();
    }

    // Name property and ValueChanged event omitted for brevity

    private Lazy<TValue> m_value;
    public TValue Value
    {
        get
        {
            return m_value.Value;
        }
    }

    public void RefreshProperty()
    {
        // Ensure we retrieve the value anew the next time it is requested
        this.m_value = new Lazy<TValue>(m_getValueProperty);

        OnValueChanged(new ValueChangedEventArgs());
        m_owner.RaisePropertyChanged(Name);
    }
}
```

should see a progress indicator. This is a seemingly simple scenario, but there's a lot of subtlety here to poke at.

The first part is tracking whether the object is busy; in the simple case, I can do that with a Boolean `NotifyProperty`. However, what often happens is that an object can be “busy” for one of multiple reasons: let's say I'm loading several areas of data, possibly in parallel. The overall “busy” state should depend on whether any of those items are still in progress. That almost sounds like a job for derived properties, but it would be clunky (if not impossible): I'd need a property for each possible operation to track whether it's in progress. Instead, I want to do something like the following for each operation, using a single `IsBusy` property:

```
try
{
    IsBusy.SetValue(true);
    await LongRunningOperation();
}
finally
{
    IsBusy.SetValue(false);
}
```

To enable this, I create an `IsBusyNotifyProperty` class that extends `NotifyProperty<bool>`, and in it, keep a “busy count.” I override `SetValue` such that `SetValue(true)` increases that count, and `SetValue(false)` decreases it. When the count goes from 0 to 1, only then do I call `base.SetValue(true)`, and when it goes from 1 to 0, `base.SetValue(false)`. In this way, starting multiple outstanding operations results in `IsBusy` becoming true only once, and thereafter it becomes false again only when they're all finished. You can see the implementation in the code download.

That takes care of the “busy” side of things: I can bind “is busy” to the visibility of a progress indicator. However, for disabling the UI, I need the opposite. When “is busy” is true, “UI enabled” should be false.

XAML has the concept of an `IValueConverter`, which converts a value to (or from) a display representation. A ubiquitous example is `BooleanToVisibilityConverter`—in XAML, an element's “Visibility” isn't described by a Boolean, but rather an enum value. This means it's not possible to bind an element's visibility directly to a Boolean property (like `IsBusy`); you need to bind the value and also use a converter. For example:

```
<StackPanel Visibility="{Binding IsBusy,
    Converter={StaticResource BooleanToVisibilityConverter}}" />
```

I mentioned that “enable the UI” is the opposite of “is busy”; it might be tempting to create a value converter to invert a Boolean property, and use that to do the job:

```
<Grid IsEnabled="{Binding IsBusy,
    Converter={StaticResource BooleanToInverseConverter}}" />
```

Indeed, before I created a `DerivedNotifyProperty` class, that was the easiest way. It was quite tedious to create a separate property, wire it up to be the inverse of `IsBusy`, and raise the appropriate `PropertyChanged` event. Now, however, it's trivial, and without that artificial barrier (that is, laziness) I have a better sense of where it makes sense to use `IValueConverter`.

Ultimately, the view—however it might be implemented (WPF or Windows Forms, for example; or even a console app is a type of view)—should be a visualization (or “projection”) of what's happening in the underlying application, and have no responsibility for determining the mechanism and business rules for what's going on. In this case, the fact that `IsBusy` and `IsEnabled` happen to be related to each other so

intimately is an implementation detail; it's not inherent that disabling the UI should be related specifically to whether the application is busy.

As it stands, I consider it a gray area, and wouldn't argue with you if you did want to use a value converter to implement this. However, I can make a much stronger case by adding another piece to the example. Let's pretend that if it loses network access, the application should also disable the UI (and show a panel indicating the situation). Well, that makes three situations: If the application is busy, I should disable the UI (and show a progress panel). If the application loses network access, I should also disable the UI (and show a “lost connection” panel). The third situation is when the application is connected and not busy and, thus, ready to accept input.

Ultimately, the view should only be a visualization of what's happening in the underlying application.

Trying to implement this without a separate `IsEnabled` property is awkward at best; you could use a `MultiBinding`, but that's still ungainly, and not supported in all environments. Ultimately, that kind of awkwardness usually means there's a better way, and now we know there is: this logic is better handled inside the view model. It's now trivial to expose two `NotifyProperties`, `IsBusy` and `IsDisconnected`, and then create a `DerivedNotifyProperty`, `IsEnabled`, that's true only if both of those are false.

If you went the `IValueConverter` route and bound the UI's `Enabled` state directly to `IsBusy` (with a converter to invert it), you'd have quite a bit of work to do now. If you instead exposed a separate, derived `IsEnabled` property, adding this new bit of logic is much less work, and the `IsEnabled` binding itself wouldn't even need to change. That's a good sign you're doing things right.

## Wrapping Up

Laying out this framework was a bit of a journey, but the reward is that now I can implement property change notifications without repetitive boilerplate, without magic strings and with support for refactoring. My view models don't require logic from a particular base class. I can create derived properties that also raise the appropriate change notifications without much additional effort. Finally, the code that I see is the code that's running. And I get all of that by developing a fairly simple framework with object-oriented design. I hope you find it useful in your own projects. ■

---

**MARK SOWUL** has been a devoted .NET developer since the beginning, and shares his wealth of architecture and performance expertise in the Microsoft .NET Framework and SQL Server via his New York consulting business, SolSoft Solutions. Reach him at [mark@solsoftsolutions.com](mailto:mark@solsoftsolutions.com). If you find his ideas intriguing and would like to subscribe to his newsletter, sign up at [eepurl.com/\\_k7YD](http://eepurl.com/_k7YD).

---

**THANKS** to the following technical experts for reviewing this article: Francis Cheung (Microsoft) and Charles Malm (Zebra Technologies)





# BEST FILE APIs

Open Create Convert Print Save

files from your *applications!*



XLS

DOC



PDF

Contact Us:

US: +1 888 277 6734

EU: +44 141 416 1112

AU: +61 2 8003 5926

[sales@aspose.com](mailto:sales@aspose.com)

SCAN FOR  
20% SAVINGS!



# BUSINESS FILE FORMATS



## ASPOSE.Cells

XLS, CSV, PDF, SVG, HTML, PNG  
BMP, XPS, JPG, SpreadsheetML...

## ASPOSE.Words

DOC, RTF, PDF, HTML, PNG  
ePUB, XML, XPS, JPG...

## ASPOSE.Pdf

PDF, XML, XSL-FO, HTML, BMP  
JPG, PNG, ePUB...

## ASPOSE.Slides

PPT, POT, POTX, XPS, HTML  
PNG, PDF...

## ASPOSE.BarCode

JPG, PNG, BMP, GIF, TIF, WMF  
ICON...

## ASPOSE.Tasks

XML, MPP, SVG, PDF, TIFF  
PNG...

## ASPOSE.Email

MSG, EML, PST, EMLX  
OST, OFT...

## ASPOSE.Imaging

PDF, BMP, JPG, GIF, TIFF  
PNG...

+ MANY MORE!

Get your FREE evaluation copy at [www.aspose.com](http://www.aspose.com)

.NET

Java

Cloud

# Working with Local Databases in Xamarin.Forms Using SQLite

Alessandro Del Sole

**More often than not**, applications work with data. This is true not only for desktop and Web applications, but also for mobile apps. In many cases, mobile apps exchange data over networks and take advantage of cloud storage and services such as push notifications. However, there are situations in which mobile apps only need to store data locally. With simple, unstructured data, such as user settings and options, applications can store information inside local files, such as XML or text, or through specific objects offered by the various development platforms. In the case of complex, structured data, applications need a different way to store information.

The good news is that you can easily include local databases in your mobile app using SQLite ([sqlite.org](http://sqlite.org)). SQLite is an open source, lightweight, serverless database engine that makes it simple to

create local databases and perform operations on data. Information is stored inside tables and data operations can be performed writing C# code and LINQ queries. SQLite perfectly suits cross-platform development, because it's a portable database engine. In fact, it's pre-installed on both iOS and Android, and it can be easily deployed to Windows, as well. For this reason, SQLite also is the perfect companion to build cross-platform, data-centric mobile apps with Xamarin.Forms that need a local database. In this article, I'll show how to create a mobile app that targets Android, iOS, and the Universal Windows Platform (UWP) with Xamarin.Forms,

Once you've created the project,  
you need a managed way to  
access SQLite databases.

and that takes advantage of SQLite to store and retrieve local data. I assume you already know how to create a Xamarin.Forms application with Visual Studio 2015; what XAML is; and how to debug a Xamarin.Forms app using the different emulators included with the various platform SDKs. For further information, you can read the following articles: "Build a Cross-Platform UX with Xamarin.Forms" ([msdn.com/magazine/mt595754](http://msdn.com/magazine/mt595754)), "Share UI Code Across Mobile Platforms with Xamarin.Forms" ([msdn.com/magazine/dn904669](http://msdn.com/magazine/dn904669)) and "Build a Cross-Platform, Mobile Golf App Using C# and

## This article discusses:

- Creating a Xamarin.Forms cross-platform app with Visual Studio 2015
- Using SQLite as a local database
- Implementing read/write operations over data
- Data binding to the UI

## Technologies discussed:

SQLite, Xamarin.Forms, C#, Visual Studio 2015

## Code download available at:

[msdn.com/magazine/0716magcode](http://msdn.com/magazine/0716magcode)



Xamarin” (msdn.com/magazine/dn630648). The latter describes how to work with data over the Microsoft Azure platform. Both this article and the sample code are based on Xamarin.Forms 2.0, which you get by installing Xamarin 4.0.3.

## Enabling SQLite on UWP Apps

The SQLite core engine is already included on iOS and Android, but not on Windows. Therefore, you need to include SQLite binaries with your app package. Instead of manually including such binaries

with every project, you can take advantage of the SQLite extension for Visual Studio 2015, which provides precompiled binaries for the database engine and automates the task of including the required files with new projects. I describe this before showing how to create new projects because the extension works at the IDE level, not at the project level, and will supply the precompiled SQLite binaries every time you include the SQLite libraries in your solutions. There are several SQLite extensions, each targeting a specific Windows version, that can be downloaded using the Extensions

and Updates tool in Visual Studio 2015, as shown in **Figure 1**.

In this case, download and install the SQLite for Universal Windows Platform extension. By doing so, a UWP app that uses SQLite will also include the precompiled database engine binaries. If required, restart Visual Studio 2015 after installing the extension.

## Creating a Sample Project

The first thing to do is create a new project based on Xamarin Forms. The project template you use in Visual Studio 2015 is called Blank App (Xamarin.Forms Portable) and is located inside the Cross-Platform folder of the Visual C# node in the New Project dialog (see **Figure 2**).

The reason for choosing the Portable project type instead of the Shared type is that you might want to produce a reusable data access layer within a library, whereas a Shared project’s scope is only within the solution to which it belongs. At the end of the article I’ll explain more thoroughly the differences between portable libraries and the Shared projects.

When you click OK, Visual Studio 2015 generates a new solution that contains projects targeting iOS, Android, UWP, Windows Runtime and Windows Phone, plus a Portable Class Library (PCL) project. The latter is where you’ll write most of the code that will be shared across the platform-specific projects.

## Installing the SQLite NuGet Package

Once you’ve created the project, you need a managed way to access SQLite databases. There are many

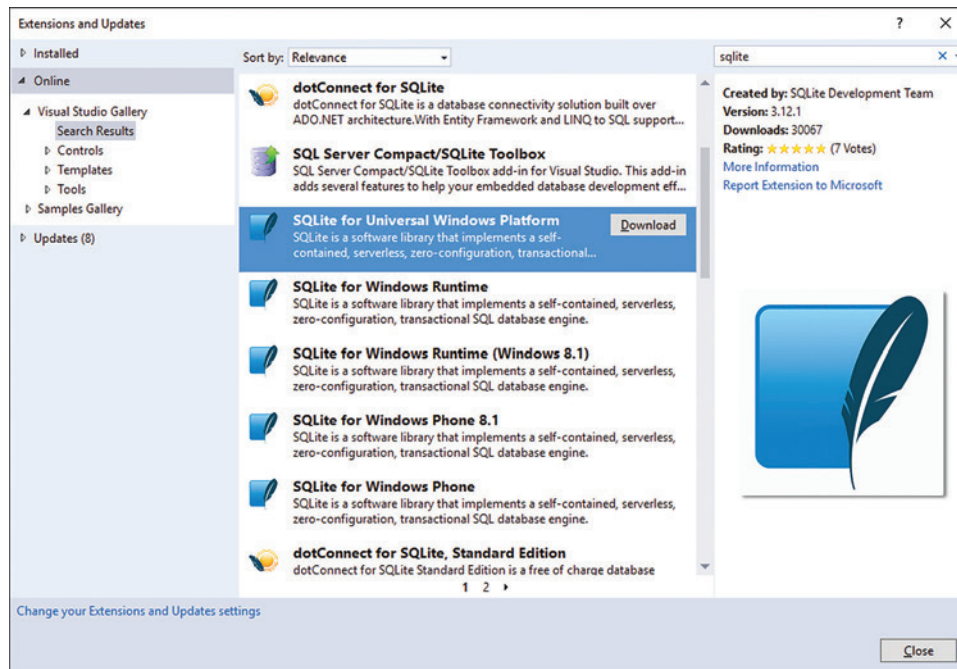


Figure 1 Downloading the SQLite for Universal Windows Platform Extension in Visual Studio 2015

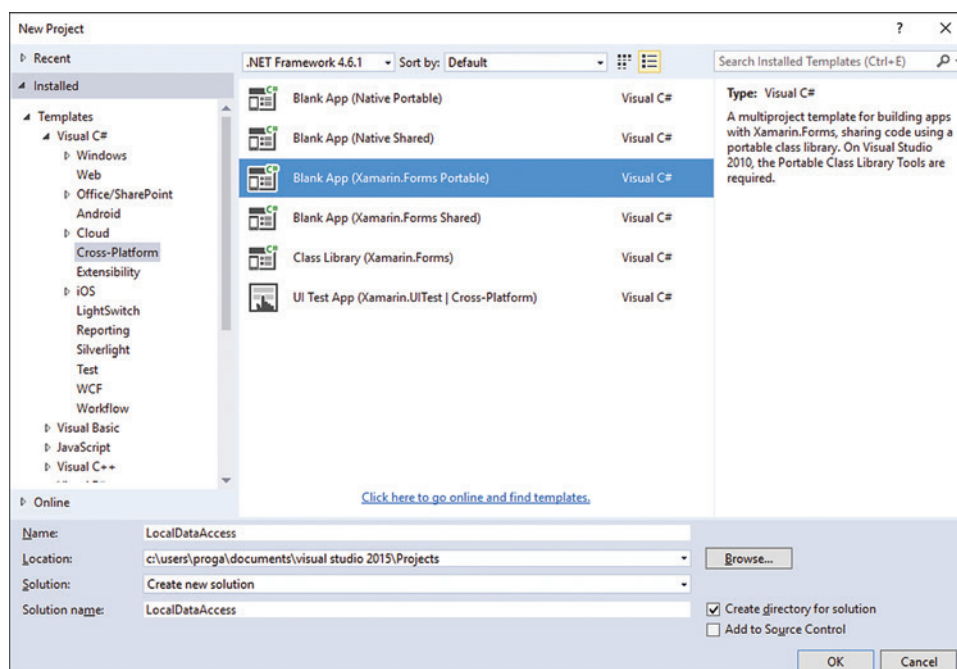


Figure 2 Creating a New Xamarin Forms Project in Visual Studio 2015

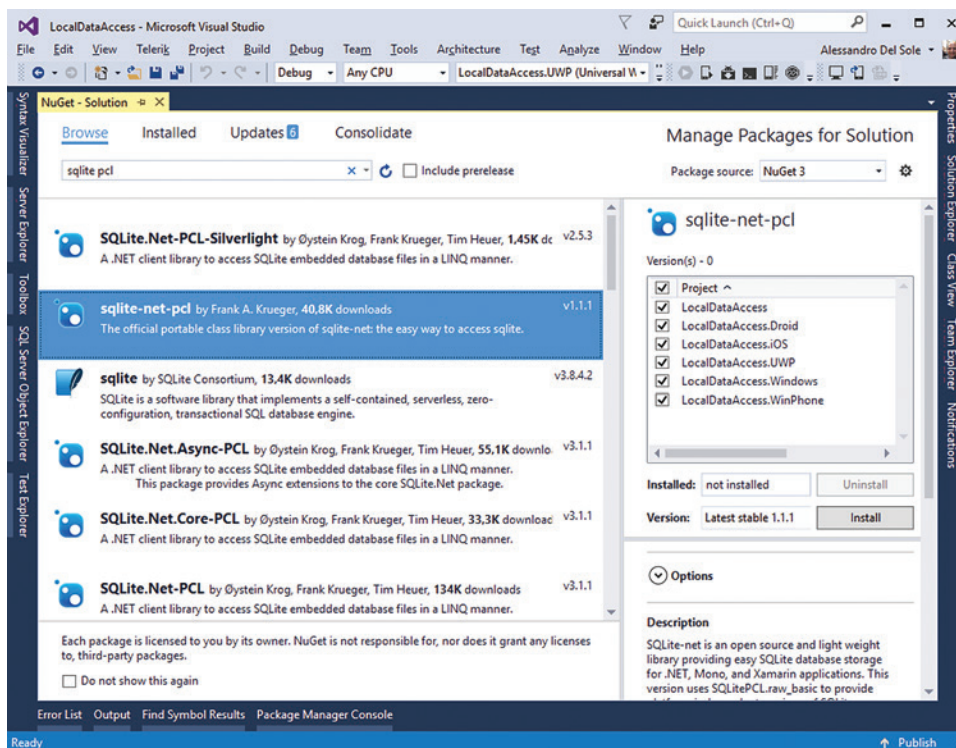


Figure 3 Installing the Proper NuGet Packages

libraries that allow working against SQLite databases in the Microsoft .NET Framework, but the one you need is a special portable library that also targets Xamarin apps. It's called SQLite-net, and it's an open source and lightweight library for .NET, Mono, and Xamarin applications. It's available as a NuGet package with the name `sqlite-net-pcl`. You can install the NuGet package at the solution level from either the NuGet Package Manager console, by typing `install sqlite-net-pcl`, or from the NuGet UI in Visual Studio 2015, which you enable by right-clicking the solution name in Solution Explorer and then selecting Manage NuGet Packages for Solution. **Figure 3** shows how to locate and install the `sqlite-net-pcl` package via the NuGet UI.

Now you have everything you need and you're ready to start coding.

## Platform-Specific Code: Supplying the Connection String

As with any kind of database, your code accesses a SQLite database through the connection string, which is the first thing you need to build. Because a SQLite database is a file that resides in a local folder, constructing the connection string requires the database pathname. Though most of the code you'll write is shared across different platforms, the way Android, iOS and Windows handle pathnames is different, so building the connection string requires platform-specific code. You then invoke the connection string via dependency injection.

In the Portable project, add a new interface called `IDatabaseConnection.cs` and write the following code:

```
public interface IDatabaseConnection
{
    SQLite.SQLiteConnection DbConnection();
}
```

This interface exposes a method called `DbConnection`, which will be implemented in each platform-specific project and will return the proper connection string.

The next step is adding a class to each platform-specific project that implements the interface and returns the proper connection string, based on a sample database I'll call `CustomersDb.db3`. (If you're not familiar with SQLite, `.db3` is the file extension that identifies SQLite databases.) In the `LocalDataAccess.Droid` project, add a new class called `DatabaseConnection_Android.cs` and write the code shown in **Figure 4**.

An attribute called `Xamarin.Forms.Dependency` indicates that the specified class implements a necessary interface. This attribute is applied at the namespace level with the assembly keyword. On Android, the database file must

be stored inside the `Personal` folder, so the database pathname is made of the filename (`CustomersDb.db3`) and the `Personal` folder path. The resulting pathname is assigned as a parameter to the constructor of the `SQLiteConnection` class and returned to the caller. On iOS, you use the same API, but the folder in which the SQLite database resides is `Personal/Library`.

Now, add a new class called `DatabaseConnection_iOS.cs` to the iOS project and write the code shown in **Figure 5**.

On Windows 10, the SQLite database resides in the app's local folder. The API you use to access the local folder is different from the other platforms, because you work with classes from the `Windows.Storage` namespace instead of `System.IO`. Add a new class called `DatabaseConnection_UWP.cs` to the Universal Windows project, and write the code shown in **Figure 6**.

This time the app's local folder path is returned by the `Windows.Storage.ApplicationData.Current.LocalFolder.Path`

Figure 4 Generating a Connection String in the Android Project

```
using SQLite;
using LocalDataAccess.Droid;
using System.IO;
[assembly: Xamarin.Forms.Dependency(typeof(DatabaseConnection_Android))]
namespace LocalDataAccess.Droid
{
    public class DatabaseConnection_Android : IDatabaseConnection
    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CustomersDb.db3";
            var path = Path.Combine(System.Environment.
                GetFolderPath(System.Environment.
                    SpecialFolder.Personal), dbName);
            return new SQLiteConnection(path);
        }
    }
}
```



# DEVELOPED FOR INTUITIVE USE

## DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.



**DynamicPDF**

[WWW.DYNAMICPDF.COM](http://www.DynamicPDF.com)



**TRY OUR PDF SOLUTIONS FREE TODAY!**

[www.DynamicPDF.com/eval](http://www.DynamicPDF.com/eval) or call 800.631.5006 | +1 410.772.8620

 **ceTe software**

Figure 5 Generating a Connection String in the iOS Project

```
using LocalDataAccess.iOS;
using SQLite;
using System;
using System.IO;

[assembly: Xamarin.Forms.Dependency(typeof(DatabaseConnection_iOS))]
namespace LocalDataAccess.iOS
{
    public class DatabaseConnection_iOS
    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CustomersDb.db3";
            string personalFolder =
                System.Environment.
                GetFolderPath(Environment.SpecialFolder.Personal);
            string libraryFolder =
                Path.Combine(personalFolder, "..", "Library");
            var path = Path.Combine(libraryFolder, dbName);
            return new SQLiteConnection(path);
        }
    }
}
```

property, which is combined with the database name to return the connection string via the `SQLiteConnection` object. Now you've written platform-specific code that allows the generation of the proper connection string based on the platform on which the app is running. From now on, all your code will be shared. The next step is implementing a data model.

## Writing a Data Model

The goal of the app is to work with a simplified list of customers stored inside a SQLite database and to support operations over data. The first thing that's needed at this point is a class to represent a customer that will be mapped to a table in the database. In the Portable project, add a class called `Customer.cs`. This class must implement the `INotifyPropertyChanged` interface to notify callers of changes in the data it stores. It will use special attributes in the SQLite namespace to annotate properties with validation rules and other information in a way that's very close to the data annotations from the `System.ComponentModel.DataAnnotations` namespace. **Figure 7** shows the sample `Customer` class.

The key point of this class is annotating objects with SQLite attributes. The `Table` attribute allows assigning a table name, in this

Figure 7 Implementing a Data Model

```
using SQLite;
using System.ComponentModel;

namespace LocalDataAccess
{
    [Table("Customers")]
    public class Customer: INotifyPropertyChanged
    {
        private int _id;
        [PrimaryKey, AutoIncrement]
        public int Id
        {
            get
            {
                return _id;
            }

            set
            {
                this._id = value;
                OnPropertyChanged(nameof(Id));
            }
        }

        private string _companyName;
        [NotNull]
        public string CompanyName
        {
            get
            {
                return _companyName;
            }

            set
            {
                this._companyName = value;
                OnPropertyChanged(nameof(CompanyName));
            }
        }

        private string _physicalAddress;
        [MaxLength(50)]
        public string PhysicalAddress
        {
            get
            {
                return _physicalAddress;
            }

            set
            {
                this._physicalAddress=value;
                OnPropertyChanged(nameof(PhysicalAddress));
            }
        }

        private string _country;
        public string Country
        {
            get
            {
                return _country;
            }

            set
            {
                _country = value;
                OnPropertyChanged(nameof(Country));
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        private void OnPropertyChanged(string propertyName)
        {
            this.PropertyChanged?.Invoke(this,
                new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Figure 6 Generating a Connection String in the Universal Windows Project

```
using SQLite;
using Xamarin.Forms;
using LocalDataAccess.UWP;
using Windows.Storage;
using System.IO;

[assembly: Dependency(typeof(DatabaseConnection_UWP))]
namespace LocalDataAccess.UWP
{
    public class DatabaseConnection_UWP : IDatabaseConnection
    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CustomersDb.db3";
            var path = Path.Combine(ApplicationData.
                Current.LocalFolder.Path, dbName);
            return new SQLiteConnection(path);
        }
    }
}
```



case Customers. This isn't mandatory, but if you don't specify one, SQLite generates a new table based on the class name, Customer in this case. So, for the sake of consistency, the code generates a new table with a plural name. The PrimaryKey and AutoIncrement attributes applied to the Id property make this the primary key in the Customers table with automatic increment. The NotNull attribute applied to the CompanyName property marks this as required, which implies that the data store validation will fail if the property value is null. The MaxLength attribute applied to the PhysicalAddress property indicates the maximum length for the property value. Another interesting attribute is Column, which you can apply to a property name to provide a different column name in the database.

## Implementing Data Access

After writing a simple data model, you need a class that provides methods that perform operations over data. For the sake of clarity and because this is a simple example, I won't use a Model-View-View-Model (MVVM) approach here; not all readers have experience with this pattern and, in any case, it's better used in large projects. You can certainly rewrite the sample as you like.

Let's start with a new class called CustomersDataAccess.cs in the Portable project, which requires the following using directives:

```
using SQLite;
using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;
using System.Collections.ObjectModel;
```

The first things you need in this class are a private field that stores the connection string and an object that will be used to implement locks on data operations, to avoid database collisions:

```
private SQLiteConnection database;
private static object collisionLock = new object();
```

More specifically, locks should have the following form:

```
// Use locks to avoid database collisions
lock(collisionLock)
{
    // Data operation here ...
}
```

With Xamarin.Forms, you use XAML to build the UI and you can take advantage of data binding to display and enter information; for this reason, you need to expose the data in a way XAML can work with it. The best approach is exposing a property of type ObservableCollection<Customer> like this:

```
public ObservableCollection<Customer> Customers { get; set; }
```

The ObservableCollection type has built-in support for change notification; therefore, it's the most appropriate collection for data binding in XAML-based platforms.

Now it's time to implement the class constructor, which is also responsible for invoking the platform-specific implementation of the DbConnection method, which returns the proper connection string, as shown in **Figure 8**.

Notice how the code invokes the DependencyService.Get method. This is a generic method that returns the platform-specific implementation of the supplied generic type, IDatabaseConnection in this case. With this approach based on dependency injection, the code is invoking the platform-specific implementation of the DbConnection method. The Xamarin runtime then knows how to resolve the method call based on the OS on which the app is

**Figure 8 Implementing the Class Constructor**

```
public CustomersDataAccess()
{
    database =
        DependencyService.Get<IDatabaseConnection>().
        DbConnection();
    database.CreateTable<Customer>();

    this.Customers =
        new ObservableCollection<Customer>(database.Table<Customer>());

    // If the table is empty, initialize the collection
    if (!database.Table<Customer>().Any())
    {
        AddNewCustomer();
    }
}
```

running. Invoking this method will also cause the database to be created if one isn't found. The SQLiteConnection object exposes a generic method called CreateTable<T>, where the generic type is your model class, Customer in this case. With this simple line of code, you create a new Customers table. If the table already exists, it won't be overwritten. The code also initializes the Customers property; it invokes the Table<T> generic method from SQLiteConnection, where the generic type is still your model class. Table<T> returns an object of type TableQuery<T>, which implements the IEnumerable<T> interface and can also be queried with LINQ. The actual returned result consists of a list of <T> objects; however, binding a TableQuery object to the UI directly is not the proper way to present data, so a new ObservableCollection<Customer> based on the returned result is generated and assigned to the Customers property. The code also invokes a method called AddNewCustomer if the table is empty, defined as follows:

```
public void AddNewCustomer()
{
    this.Customers.
        Add(new Customer
        {
            CompanyName = "Company name...",
            PhysicalAddress = "Address...",
            Country = "Country..."
        });
}
```

This method simply adds a new Customer to the Customers collection with default property values and avoids binding to an empty collection.

**Figure 9 Filtering a List of Customers by Country**

```
public IEnumerable<Customer> GetFilteredCustomers(string countryName)
{
    lock(collisionLock)
    {
        var query = from cust in database.Table<Customer>()
                     where cust.Country == countryName
                     select cust;
        return query.AsEnumerable();
    }
}

public IEnumerable<Customer> GetFilteredCustomers()
{
    lock(collisionLock)
    {
        return database.Query<Customer>(
            "SELECT * FROM Item WHERE Country = 'Italy'").AsEnumerable();
    }
}
```

Figure 10 Insert or Update a Single Instance of a Customer Object Depending on the Existence of a Customer Class ID

```
public int SaveCustomer(Customer customerInstance)
{
    lock(collisionLock)
    {
        if (customerInstance.Id != 0)
        {
            database.Update(customerInstance);
            return customerInstance.Id;
        }
        else
        {
            database.Insert(customerInstance);
            return customerInstance.Id;
        }
    }
}
```

## Querying Data

Querying data is very important. SQLite has essentially two ways of implementing queries. The first is using LINQ against the result of an invocation to the `Table<T>` method, which is an object of type `TableQuery<T>` and implements the `IEnumerable<T>` interface. The second is invoking a method called `SQLiteConnection.Query<T>`, which takes an argument of type strings that represents a query written in SQL. The code in **Figure 9** demonstrates how to filter the list of customers by country, using both approaches.

The first overload of `GetFilteredCustomers` returns the result of a LINQ query that filters data based on the country name, supplied as the method argument. The second overload invokes `Query` to execute SQL queries directly. This method expects the result to be a generic List, whose generic type is the same passed to `Query`. A `SQLiteException` is thrown if the query fails. Of course, you can retrieve a specified object instance using LINQ or extension methods, as in the following code that invokes `FirstOrDefault` over the list of customers and retrieves the specified customer instance based on its id:

```
public Customer GetCustomer(int id)
{
    lock(collisionLock)
    {
        return database.Table<Customer>().
            FirstOrDefault(customer => customer.Id == id);
    }
}
```

## Executing CRUD Operations

Create, read, update and delete (CRUD) operations are also extremely important. Reading data is typically performed using the approaches described in the previous section, so now I'll discuss how to create, update and delete information in a SQLite database. The `SQLiteConnection` object exposes the `Insert`, `InsertAll`, `Update`, and `UpdateAll` methods to insert or update objects in the database. `InsertAll` and `UpdateAll` execute an insert or update operation on a collection that implements `IEnumerable<T>` passed as the argument. The insert or update operation is executed in a batch, and both methods also allow executing the operation as a transaction. Keep in mind that while `InsertAll` requires that no items in the collection exist in the database, `UpdateAll` requires that all the items in the collection already exist in the database. In this case, I have an `ObservableCollection<Customer>` that can contain both objects retrieved from the database and new objects added via the UI that haven't been saved yet, or pending edits

Figure 11 Insert or Update All Instances of Customer

```
public void SaveAllCustomers()
{
    lock(collisionLock)
    {
        foreach (var customerInstance in this.Customers)
        {
            if (customerInstance.Id != 0)
            {
                database.Update(customerInstance);
            }
            else
            {
                database.Insert(customerInstance);
            }
        }
    }
}
```

over existing objects. For this reason, using `InsertAll` and `UpdateAll` isn't recommended. A good approach is simply checking if an instance of the `Customer` class has an `Id`. If it does, the instance already exists in the database so it just needs to be updated. If the `Id` is zero, the instance doesn't exist in the database; therefore, it must be saved. The code in **Figure 10** demonstrates how to insert or update a single instance of a `Customer` object based on the previous consideration.

The code in **Figure 11**, instead, demonstrates how to insert or update all instances of `Customer`.

Both the `Insert` and `Update` methods return an integer that represents the number of rows added or updated. `Insert` also offers an overload that accepts a string containing additional SQL statements you might want to execute against the inserted rows. Also, it's worth mentioning that `Insert` automatically updates, by reference, the property in your business object that has been mapped to be the primary key, `Customer.Id` in this case. The `SQLiteConnection` class also exposes the `Delete<T>` and `DeleteAll<T>` methods, which permanently delete one or all objects from a table. The delete operation is irreversible, so be aware of what you're doing. The following code implements a method called `DeleteCustomer` that deletes the specified customer instance from both the in-memory `Customers` collection and the database:

```
public int DeleteCustomer(Customer customerInstance)
{
    var id = customerInstance.Id;
    if (id != 0)
    {
        lock(collisionLock)
        {
            database.Delete<Customer>(id);
        }
    }
    this.Customers.Remove(customerInstance);
    return id;
}
```

If the specified `Customer` has an `id`, it exists in the database, so it's permanently deleted, and it's also removed from the `Customers` collection. `Delete<T>` returns an integer that represents the number of deleted rows. You can also permanently delete all objects from a table. You can certainly invoke `DeleteAll<T>`, where the generic type is your business object such as `Customer`, but I want to show an alternative approach instead, so you can get knowledge of other members. The `SQLiteConnection` class exposes a method called `DropTable<T>`, which permanently destroys a table in the database. For instance, you might implement a table deletion as follows:



## Help & Manual Professional | from \$586.04



Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePUB, Kindle or print
- Styles and Templates give you full design control

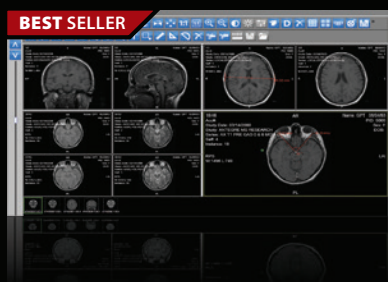


## Aspose.Total for .NET | from \$2,449.02



Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR and OneNote files alongside many more document management features in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types

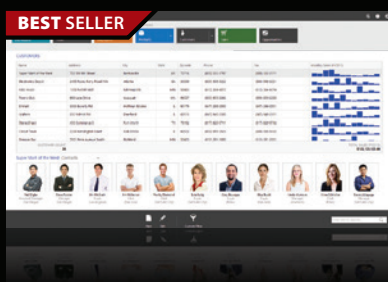


## LEADTOOLS Medical Imaging SDKs V19 | from \$4,995.00 SRP



Powerful DICOM, PACS, and HL7 functionality.

- Load, save, edit, annotate & display DICOM Data Sets with support for the latest specifications
- High-level PACS Client and Server components and frameworks
- OEM-ready HTML5 Zero-footprint Viewer and DICOM Storage Server apps with source code
- Medical-speci-fic image processing functions for enhancing 16-bit grayscale images
- Native libraries for .NET, C/C++, HTML5, JavaScript, WinRT, iOS, OS X, Android, Linux, & more



## DevExpress DXperience 15.2 | from \$1,439.99



The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

- WinForms Grid: New data-aware Tile View
- WinForms Grid & TreeList: New Excel-inspired Conditional Formatting
- .NET Spreadsheet: Grouping and Outline support
- ASP.NET: New Rich Text Editor-Word Processing control
- ASP.NET Reporting: New Web Report Designer

```

public void DeleteAllCustomers()
{
    lock(collisionLock)
    {
        database.DropTable<Customer>();
        database.CreateTable<Customer>();
    }
    this.Customers = null;
    this.Customers = new ObservableCollection<Customer>
        (database.Table<Customer>());
}

```

The code deletes the Customers table, then creates one, and finally cleans up and recreates the Customers collection. **Figure 12** shows the full listing for the CustomersDataAccess.cs class.

## A Simple UI with Data Binding

Now that you have the model and the data access layer, you need a UI to present and edit data. As you know, with Xamarin.Forms the

Figure 12 The CustomersDataAccess.cs Class

```

using SQLite;
using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;
using System.Collections.ObjectModel;

namespace LocalDataAccess
{
    public class CustomersDataAccess
    {
        private SQLiteConnection database;
        private static object collisionLock = new object();
        public ObservableCollection<Customer> Customers { get; set; }

        public CustomersDataAccess()
        {
            database =
                DependencyService.Get<IDatabaseConnection>().
                DbConnection();
            database.CreateTable<Customer>();

            this.Customers =
                new ObservableCollection<Customer>(database.Table<Customer>());

            // If the table is empty, initialize the collection
            if (!database.Table<Customer>().Any())
            {
                AddNewCustomer();
            }
        }

        public void AddNewCustomer()
        {
            this.Customers.
                Add(new Customer
                {
                    CompanyName = "Company name...",
                    PhysicalAddress = "Address...",
                    Country = "Country..."
                });
        }

        // Use LINQ to query and filter data
        public IEnumerable<Customer> GetFilteredCustomers(string countryName)
        {
            // Use locks to avoid database collisions
            lock(collisionLock)
            {
                var query = from cust in database.Table<Customer>()
                    where cust.Country == countryName
                    select cust;
                return query.AsEnumerable();
            }
        }

        // Use SQL queries against data
        public IEnumerable<Customer> GetFilteredCustomers()
        {
            lock(collisionLock)
            {
                return database.
                    Query<Customer>
                    ("SELECT * FROM Item WHERE Country = 'Italy'").AsEnumerable();
            }
        }

        public Customer GetCustomer(int id)
        {
            lock(collisionLock)
            {
                return database.Table<Customer>().
                    FirstOrDefault(customer => customer.Id == id);
            }
        }

        public int SaveCustomer(Customer customerInstance)
        {
            lock(collisionLock)
            {
                if (customerInstance.Id != 0)
                {
                    database.Update(customerInstance);
                    return customerInstance.Id;
                }
                else
                {
                    database.Insert(customerInstance);
                    return customerInstance.Id;
                }
            }
        }

        public void SaveAllCustomers()
        {
            lock(collisionLock)
            {
                foreach (var customerInstance in this.Customers)
                {
                    if (customerInstance.Id != 0)
                    {
                        database.Update(customerInstance);
                    }
                    else
                    {
                        database.Insert(customerInstance);
                    }
                }
            }
        }

        public int DeleteCustomer(Customer customerInstance)
        {
            var id = customerInstance.Id;
            if (id != 0)
            {
                lock(collisionLock)
                {
                    database.Delete<Customer>(id);
                }
                this.Customers.Remove(customerInstance);
                return id;
            }
        }

        public void DeleteAllCustomers()
        {
            lock(collisionLock)
            {
                database.DropTable<Customer>();
                database.CreateTable<Customer>();
            }
            this.Customers = null;
            this.Customers = new ObservableCollection<Customer>
                (database.Table<Customer>());
        }
    }
}

```



# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

[vslive.com/anaheim](http://vslive.com/anaheim)

## Anaheim SEPT 26-29

HYATT REGENCY, A DISNEYLAND® GOOD NEIGHBOR HOTEL

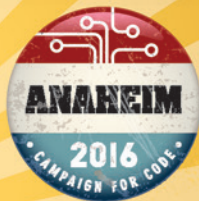


CAMPAIGN FOR CODE 2016 • VISUAL STUDIO LIVE!

# CODE

Anaheim

## FOR A BETTER TOMORROW



### DEVELOPMENT TOPICS INCLUDE:

- ALM / DevOps
- ASP.NET
- Cloud Computing
- Database and Analytics
- JavaScript/HTML5 Client
- Mobile Client
- Software Practices
- Visual Studio / .NET Framework
- Windows Client

### Register by July 20 and Save \$300

Scan the QR code to register  
or for more event details.

USE PROMO CODE VSLANTI



SUPPORTED BY



Visual Studio  
MAGAZINE

PRODUCED BY



## VSLIVE.COM/ANAHEIM



## PRACTICAL & UNBIASED TRAINING FOR DEVELOPERS:

- ALM / DevOps
- Cloud Computing
- Database and Analytics
- Mobile Client
- Software Practices
- UX / Design
- Visual Studio / .NET Framework
- Web Client
- Web Server

Register by August 3  
and Save \$300

Scan the QR code to register  
or for more event details.



USE PROMO CODE VSLDCTI



SUPPORTED BY



PRODUCED BY



VSLIVE.COM/DC

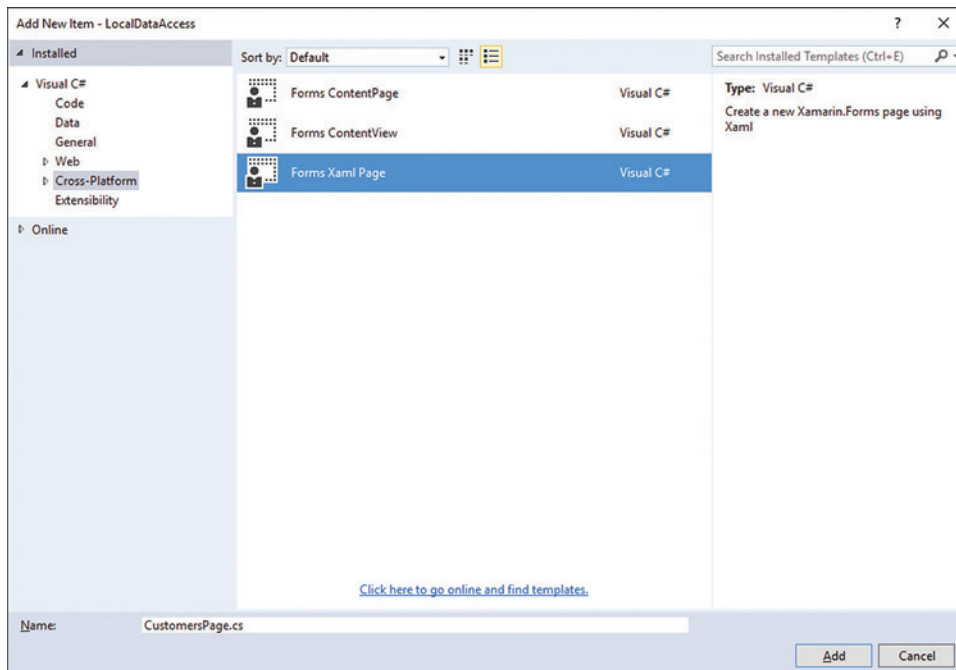


Figure 13 Adding a New Page Based on XAML

UI can be written with either C# or XAML, but the latter provides better separation between the UI and the procedural code and gives you an immediate perception of the hierarchical UI organization, so that's my choice for this article. It's worth mentioning that, with Xamarin.Forms 2.0, you can also enable XAML compilation (XamlC) for performance optimization and compile-time error check. For further details about XamlC, visit [bit.ly/24BSUC8](http://bit.ly/24BSUC8).

Figure 14 The CustomersPage User Interface

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="LocalDataAccess.CustomersPage">

  <ListView x:Name="CustomersView"
    ItemsSource="{Binding Path=Customers}"
    ListView.RowHeight="150">
    <ListView.ItemTemplate>
      <DataTemplate>
        <ViewCell>
          <StackLayout Orientation="Vertical">
            <Entry Text="{Binding Id}" IsEnabled="False"/>
            <Entry Text="{Binding CompanyName}" />
            <Entry Text="{Binding PhysicalAddress}" />
            <Entry Text="{Binding Country}" />
          </StackLayout>
        </ViewCell>
      </DataTemplate>
    </ListView.ItemTemplate>
  </ListView>

  <ContentPage.ToolbarItems>
    <ToolbarItem Name="Add" Activated="OnAddClick"
      Priority="0" Order="Secondary" />
    <ToolbarItem Name="Remove" Activated="OnRemoveClick"
      Priority="1" Order="Secondary" />
    <ToolbarItem Name="Remove all" Activated="OnRemoveAllClick"
      Priority="2" Order="Secondary" />
    <ToolbarItem Name="Save" Activated="OnSaveClick"
      Priority="3" Order="Secondary" />
  </ContentPage.ToolbarItems>
</ContentPage>
```

Now let's write a simple page that shows a list of data with some buttons. In Xamarin.Forms, pages are shared elements, so they're added to the Portable project. To do so, in Solution Explorer right-click the Portable project and select Add | New Item. In the Add New Item dialog, locate the Cross-Platform node and select the Forms Xaml Page template, as shown in **Figure 13**. Name the new page CustomersPage.cs and click Add.

In order to show the list of customers, the simple UI will be composed of a ListView control that's data bound to the Customers collection exposed by the CustomersDataAccess class. The items' DataTemplate consists of four Entry controls, each data bound to a property of the Customer model class. If you have

experience with other XAML-based platforms such as Windows Presentation Foundation (WPF) and UWP, you can consider Entry as the equivalent for the TextBox control. The Entry controls are grouped inside a StackLayout panel, which is included in a ViewCell container. For those who come from WPF and UWP, the StackLayout is the Xamarin equivalent of the StackPanel container. ViewCell allows for creating custom cells within item controls (such as ListView). You'll notice that I'm using an Entry control with the IsEnabled property assigned as False for the Customer.Id property, instead of a Label control, which is read-only by nature. As you might remember, when you invoke the SQLiteConnection.Insert method, this updates the property mapped as the primary key in your model, so the UI should be able to automatically reflect this change. Unfortunately, the Label control doesn't update itself with the new value, whereas the Entry control does, and this is the reason Entry is used, but set as read-only.

For those who come from WPF and UWP, the StackLayout is the Xamarin equivalent of the StackPanel container.

The second part of the UI consists of ToolbarItem buttons, which provide an easy and shared way to offer user interaction via a convenient menu bar that will be available on all platforms. For the sake of simplicity, these buttons will be implemented in the secondary



Figure 15 The CustomersPage Codebehind

```
using System;
using System.Linq;
using Xamarin.Forms;

namespace LocalDataAccess
{
    public partial class CustomersPage : ContentPage
    {
        private CustomersDataAccess dataAccess;
        public CustomersPage()
        {
            InitializeComponent();

            // An instance of the CustomersDataAccessClass
            // that is used for data-binding and data access
            this.dataAccess = new CustomersDataAccess();
        }

        // An event that is raised when the page is shown
        protected override void OnAppearing()
        {
            base.OnAppearing();

            // The instance of CustomersDataAccess
            // is the data binding source
            this.BindingContext = this.dataAccess;
        }

        // Save any pending changes
        private void OnSaveClick(object sender, EventArgs e)
        {
            this.dataAccess.SaveAllCustomers();
        }

        // Add a new customer to the Customers collection
        private void OnAddClick(object sender, EventArgs e)
        {
            this.dataAccess.AddNewCustomer();
        }

        // Remove the current customer
        // If it exist in the database, it will be removed
        // from there too
        private void OnRemoveClick(object sender, EventArgs e)
        {
            var currentCustomer =
                this.CustomersView.SelectedItem as Customer;
            if (currentCustomer != null)
            {
                this.dataAccess.DeleteCustomer(currentCustomer);
            }
        }

        // Remove all customers
        // Use a DisplayAlert object to ask the user's confirmation
        private async void OnRemoveAllClick(object sender, EventArgs e)
        {
            if (this.dataAccess.Customers.Any())
            {
                var result =
                    await DisplayAlert("Confirmation",
                        "Are you sure? This cannot be undone",
                        "OK", "Cancel");

                if (result == true)
                {
                    this.dataAccess.DeleteAllCustomers();
                    this.BindingContext = this.dataAccess;
                }
            }
        }
    }
}
```

area of the menu bar, which doesn't require platform-specific icons. **Figure 14** shows the full code for the UI.

Notice how each `ToolbarItem` has the `Order` property assigned as `Secondary`; if you want to make them available in the primary area of the toolbar and supply some icons, change this to `Primary`. Also, the `Priority` property allows specifying the order in which a `ToolbarItem` appears on the toolbar, while `Activated` can be compared to a click event and requires an event handler.

The next step is writing C# code that instantiates the `CustomersDataAccess` class, data binds objects, and performs operations over data. **Figure 15** shows the C# codebehind for the page (refer to the comments for more information).

The `BindingContext` property is the equivalent of `DataContext` in WPF and UWP and represents the data source for the current page.

## Testing the Application with Emulators

Now it's time to test the application. In the `App.cs` file, you need to change the startup page. When you create a `Xamarin.Forms` project, Visual Studio 2015 generates a page written in procedural code and assigned to an object called `MainPage`. This assignment is in the `App` class constructor, so open `App.cs` and replace the `App` constructor as follows:

```
public App()
{
    // The root page of your application
    MainPage = new NavigationPage(new CustomersPage());
}
```

You might be surprised that an instance of the `CustomersPage` isn't assigned to `MainPage` directly; rather, it's encapsulated as the

parameter for an instance of the `NavigationPage` class. The reason is that using a `NavigationPage` is the only way to show a menu bar on Android, but this does not at all affect UI behavior. Depending on which platform you want to test the app, select the startup project and a proper emulator in the standard toolbar in Visual Studio and press F5. **Figure 16** shows the app running on Android and on Windows 10 Mobile.

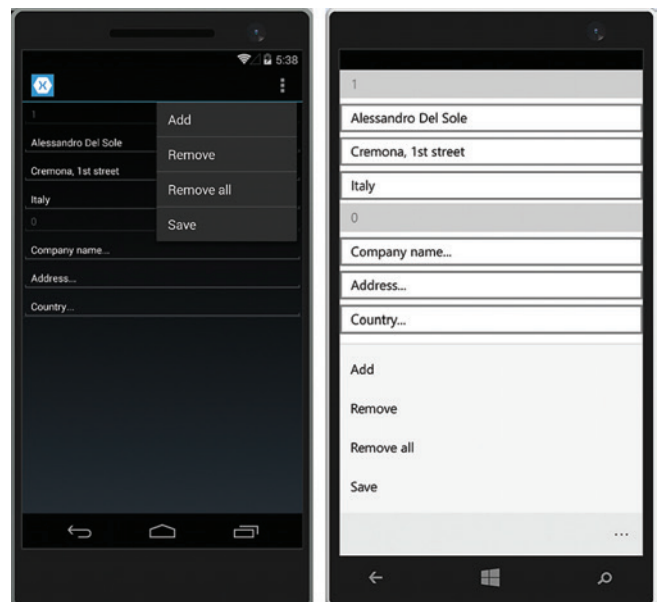


Figure 16 The Sample App Running on Different Platforms



# Microsoft Cloud Training for the Enterprise

We bring the cloud experts. You choose your pace.

## On-Demand Online Training

Choose the time, the place and the pace to fit your schedule.

## Instructor-Led Training

Engage in real time with your instructor either online or in-person for a personalized training experience. (Onsite or Virtual)

Learning Paths Include:



Microsoft  
Azure



Dynamics  
CRM Online



Microsoft  
Office 365



Visual Studio  
Team Services

Go to [Opsgility.com](https://Opsgility.com) now to start your free,  
30-day trial with code: MSDNMAG

**REAL CODE. REAL LABS. REAL LEARNING.**

[Opsgility.com](https://Opsgility.com)

The Opsgility logo, which consists of a stylized white cloud icon followed by the word "opsgility" in a lowercase, sans-serif font, all in white on a dark blue background.

Notice how the toolbar items are properly shown in the menu bar and how you can work with data in the same way on different devices and OSes.

## Writing Platform-Specific Code with Shared Projects

Sharing code is a key concept in Xamarin.Forms. In fact, all the code that doesn't leverage platform-specific APIs can be written once and shared across iOS, Android and Windows projects. When you create a Xamarin.Forms project, Visual Studio 2015 offers both the Blank App (Xamarin.Forms Portable) and the Blank App (Xamarin.Forms Shared) project templates, based on PCLs and shared projects, respectively. Generally speaking, in Visual Studio you can share code using either PCLs, which produce reusable .dll libraries that target multiple platforms but don't allow writing platform-specific code, or shared projects, which don't produce an assembly, so their scope is limited to the solution to which they belong. Shared projects do allow writing platform-specific code.

With SQLite, Xamarin.Forms applications can easily manage local databases using an open source, serverless, and portable engine that supports C# and LINQ queries.

In the case of Xamarin.Forms, when Visual Studio 2015 generates a new solution, it adds either a PCL project or a shared project depending on the template you select. In both cases, the selected template is the place you put all the shared code. But in the portable project you code interfaces that will have a platform-specific

implementation in the iOS, Android and Windows projects, whose members will then be invoked via dependency injection. This is what you've seen in this article.

In the case of shared projects, you can use conditional preprocessor directives (`#if`, `#else`, `#endif`) and environment variables that easily let you understand what platform your app is running on, so you can write the platform-specific code in the shared project directly. In the sample app described in this article, the connection string is constructed using platform-specific APIs. If you use a shared project, you could write the code shown in **Figure 17** directly in the shared project. Remember that a shared project doesn't support NuGet packages, so you must include the SQLite.cs file (available on GitHub at [bit.ly/1QU8uiR](http://bit.ly/1QU8uiR)).

As you can see, you use the `#if` and `#else` directives to detect on which platform the app is running. Each platform is represented by the `__IOS__`, `__ANDROID__` or `__WINPHONE__` environment variables, where `__WINPHONE__` targets Windows 8.x, Windows Phone 8.x and the UWP. Choosing between portable libraries and shared projects strictly depends on your needs. Portable libraries are reusable and require very clean separation between shared and platform-specific code. Shared projects let you write platform-specific code together with shared code, but they don't produce reusable libraries and they're more difficult to maintain when your code base grows very much.

## Further Improvements

The sample application described in this article can be certainly improved in many ways. For instance, you might want to implement the MVVM pattern and expose commands to the UI, instead of handling click events, and you could consider moving toolbar items to the primary area in the menu bar, supplying platform-specific icons. From the point of view of data, you might need to work with relationships and foreign keys. Because handling both with the SQLite library isn't easy, you might want to consider the SQLite-Net Extensions library ([bit.ly/24yhnpP](http://bit.ly/24yhnpP)), an open source project that simplifies the C# code you need to work with relationships and with more advanced scenarios. This is just a small list of possible improvements you could try for further studies.

## Wrapping Up

In many situations, mobile apps need local data storage. With SQLite, Xamarin.Forms applications can easily manage local databases using an open source, serverless, and portable engine that supports C# and LINQ queries. SQLite offers very intuitive objects for working with tables and database objects, making it very easy to implement local data access on any platform. Check out the SQLite documentation ([sqlite.org/docs.html](http://sqlite.org/docs.html)) for further information and additional scenarios. ■

**ALESSANDRO DEL SOLE** has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos, and articles about .NET development with Visual Studio. Del Sole works as a solution developer expert for Brain-Sys ([brain-sys.it](http://brain-sys.it)), focusing on .NET development, training and consulting. You can follow him on Twitter: @progalex.

**THANKS** to the following technical experts for reviewing this article:  
Kevin Ashley and Sara Silva

Figure 17 Writing the Connection String in a Shared Project with Conditional Preprocessor Directives

```
private string databasePath {
    get {
        var dbName = "CustomersDb.db3";
        #if __IOS__
            string folder = Environment.GetFolderPath(
                Environment.SpecialFolder.Personal);
            folder = Path.Combine(folder, "..", "Library");
            var databasePath = Path.Combine(folder, dbName);
        #else
            #if __ANDROID__
                string folder = Environment.GetFolderPath(
                    Environment.SpecialFolder.Personal);
                var databasePath = Path.Combine(folder, dbName);
            #else // WinPhone
                var databasePath =
                    Path.Combine(Windows.Storage.ApplicationData.Current.
                        LocalFolder.Path, dbName);
            #endif
        #endif

        return databasePath;
    }
}
```

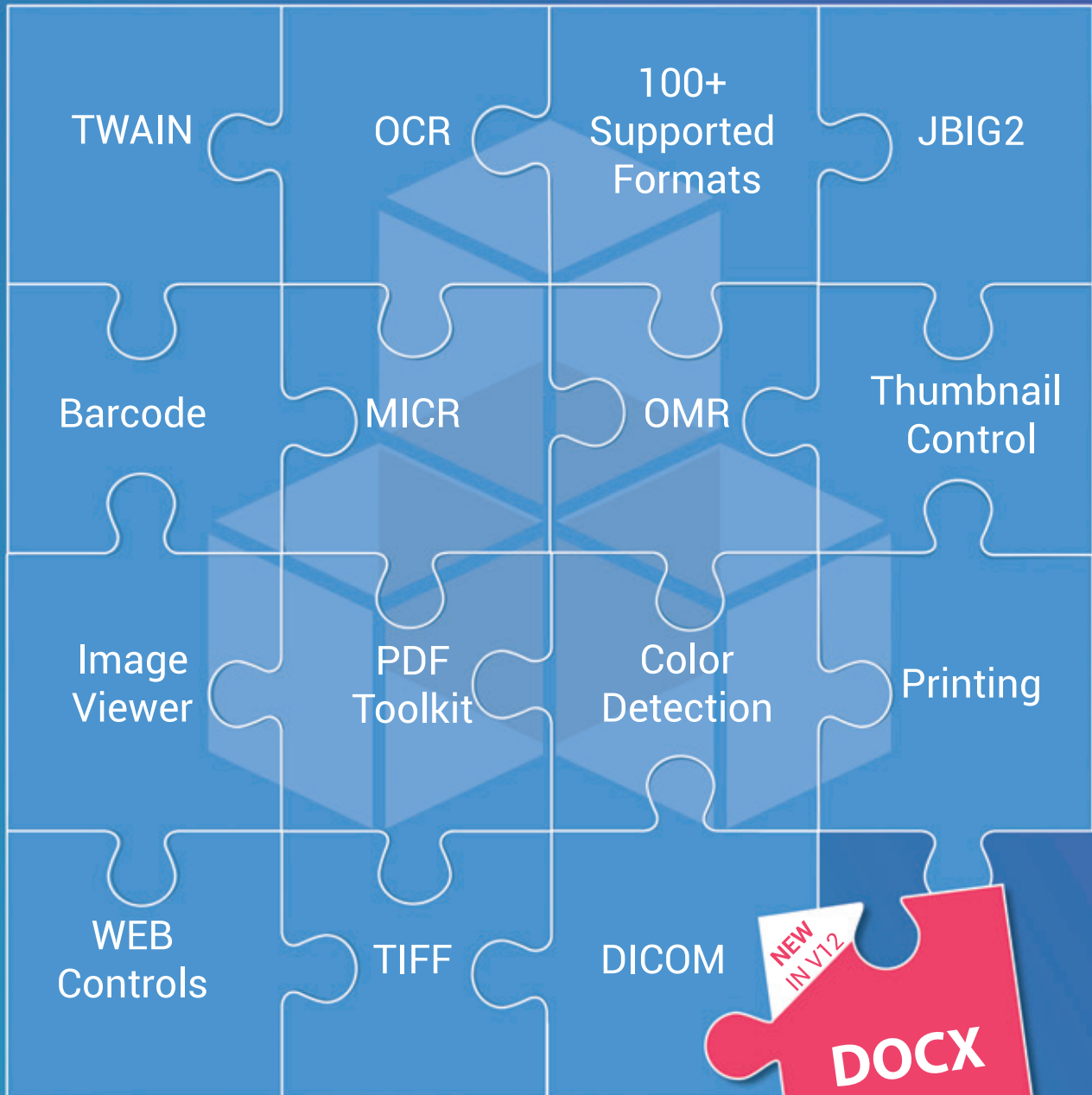
# GdPicture.NET



12

**100% ROYALTY FREE**

Imaging SDK For **WinForms**, **WPF** And **Web** Development



Try **GdPicture.NET V12** for **Free** for 30 days

**[www.gdpicture.com](http://www.gdpicture.com)**

# Leverage CQRS to Create Highly Responsive Systems

Peter Vogel

**The Command Query** Responsibility Separation (CQRS) pattern has grown in popularity over the last three to four years. Certainly, it's an essential tool in collaborative scenarios, where there's a set of data updated by multiple processes (Dino Esposito makes a case for using CQRS even more broadly in his June 2015 Cutting Edge column, "CQRS for the Common Application," at [bit.ly/10tQba3](http://bit.ly/10tQba3)). I'd go further and claim that, in fact, CQRS is the default design pattern for ASP.NET MVC developers who query data to display in their views and then issue commands to update tables when that data is posted back to their MVC controllers.

## This article discusses:

- Using DDD to segment the problem space
- Leveraging CQRS to support unique data requirements
- Implementing eventual consistency with Message Queues and RESTful services
- Integrating a command bus into processing to simplify coordinating domains

## Technologies discussed:

Domain-Driven Design (DDD), Command Query Responsibility Separation (CQRS), Relational and NoSQL Databases, Event Sourcing

## Code download available at:

[msdn.com/magazine/0716magcode](http://msdn.com/magazine/0716magcode)

However, CQRS is a tactic that should be applied as part of a larger strategy. The first step in that strategy is Domain-Driven Design (DDD), which is described in Julie Lerman's June 2013 column, "Shrink EF Models with DDD Bounded Contexts," at [bit.ly/1TfF7dk](http://bit.ly/1TfF7dk). DDD leads to breaking your application into cooperating domains, each of which may even have its own database in addition, of course, to its own dedicated business model. DDD provides strategies and tactics that let domains be developed independently of each other while still working together.

## Defining Inventory Domains

But DDD only *potentially* eliminates the need for collaboration. Consider, for example, an online sales application. Here, there is a critical shared piece of data: inventory levels. To ensure that the business doesn't try to sell something it doesn't have, the business can either keep accurate inventory counts of Quantity on Hand (QoH) for each Stock Keeping Unit (SKU) ... or always have extra, "just-in-case" inventory on hand. In today's lean world that second option isn't considered: Companies don't want to keep more inventory than they have to.

Updating QoH based on business transactions is more complicated than you might think because a real-world inventory system handles a wide variety of transactions. The obvious transactions are, of course, decrementing QoH when a SKU is sold and incrementing QoH when new SKUs are received. In addition, a company will periodically do an "inventory count" to determine the actual



QoH for each SKU. Because even in a well-managed system inventory accuracy isn't 100 percent, that count will require a change to the inventory levels. In addition, sometimes SKUs are discovered to be defective in some way and removed from inventory. Sometimes, after a SKU is sold and removed from inventory, the customer cancels the order and the SKU is returned to the shelf.

Both accounting and operations  
will need the flexibility of a  
relational database to join  
together tables in a variety of ways.

Companies want to track all of these different transactions, keeping the information specific to each transaction. When new SKUs are received, for example, companies want to know what invoice was used to buy the SKUs; when SKUs are discovered to be defective, companies want to know why; and, during inventory taking, companies want to know how big the discrepancy was. Accounting needs this information to accurately report the “state of the company”; the operations department needs this information to accurately plan for the future. Because of the need for this additional information, these transactions can't be treated as just additions and removals from inventory.

However, not all of these transactions belong in the same domain. These transactions are divided up among domains called “sales,” “accounting,” “operations,” “receiving” and so on. Dividing transactions among domains reflects the reality that different domains have different demands.

Most of the domains, for example, don't need up-to-the-minute data—if they were running even a business day behind the actual transactions, it wouldn't be a problem. The accounting department, for example, only needs to know the financial state of the inventory at month's end and, even then, might not expect to have that information until the first few days of the following month. While it might be possible to keep inventory data more current, it'd be hard to find a business justification for doing that. Those departments' inventory information can be “eventually consistent.”

The sales system can't have “eventually consistent” inventory information, though. The sales department needs to know what the QoH is right now so that it can decide whether a SKU can be shown to the customer (“Only two left! Order now!”). In fact, while most domains would have a single number for the QoH for a SKU, the sales system might keep the QoH as two numbers. One number is the “reserved” quantity (SKUs requested by a user who's in the process of creating an order) and the second number is the “still available for sale.” If a customer buys two items, the reserved number is increased by two and the available for-sale number reduced by two; at the end of the sale either the reserved quantity is reduced by two or, if the user cancels the order, added back to the available for-sale number.

Both accounting and operations will need the flexibility of a relational database to join together tables in a variety of ways. They also will need the ability to search that data, sometimes in ways that hadn't been considered prior to the discovery of a particular problem. Given the amount of data involved and the need to research the history of transactions, paging will also be required.

The sales system doesn't require as much flexibility. The relationships between entities are fixed with the design of the UI, as are the search requirements (though paging support is still required).

Response time demands also vary among domains. For most departments, a response time measured in seconds wouldn't harm the company; for the sales system, response time must be measured in fractions of a second.

Building a single system to meet all of these needs would be difficult (I'd say impossible). Building an application for each domain is, at least, possible. For example, the product management department would have a product list that's constantly being updated both with new products and information about existing products; the sales domain might, on the other hand, keep a read-only/query-only product list that's regularly synchronized with the data in the product management domain.

Think of domains as the single responsibility principle applied at the enterprise level. Each domain handles one part of the business well. While the enterprise is complicated, each domain can be—relatively speaking—simple.

## The CQRS Solution

All of these domains are still sharing the inventory levels, however. As transactions pass through domains such as accounting and receiving, they must notify the sales system of the changes to inventory levels. Even within the sales system, multiple customers might be attempting to purchase the same SKUs, individually driving stock levels up and down, and requiring some level of locking as those numbers are adjusted.

Think of domains as the single  
responsibility principle applied at  
the enterprise level.

The CQRS pattern becomes useful here by going beyond what the typical ASP.NET MVC developer would consider. Within most domains, for example, the applications can query their own databases, which contain the information that the domain needs. Once it comes time to issue a command to adjust inventory levels, all domains must update the online sales domain's data. And the obligation goes both ways: As items are sold, the sales system must notify accounting, operations and other domains about changes in QoH due to sales for each SKU.

Rather than update another domain's data, however, each domain is only obliged to notify other domains about something in which those other domains are interested (in this case, QoH). Each

domain must be responsible for updating its own data because each domain knows how to manage its data and no other domain does.

The operations domain, for example, is constantly exploring the relationships among its data to predict inventory demands and to determine what's driving stock-level fluctuations. That domain must support the flexibility in querying data that a traditional relational database provides. The complexity in the operations domain is driven by the kind of analysis required in that domain.

The sales domain, on the other hand, needs something simpler. It needs to know what the QoH (reserved and available for sale) is for any SKU. It might even make sense for the sales system to just keep the ID for every SKU and its two QoH numbers constantly in memory. If that isn't possible because of the number of inventory items, it might still make sense to keep in memory the 20 percent of the inventory that drives 80 percent of the company's sales activity. The other inventory items could be held in some NoSQL database that's designed to support sales transactions without needing to provide the flexibility that, for example, the operations domain requires. The complexity in the sales domain is driven by the need for low response times.

## The distinction between commands and events is conceptual, not technical.

These differences mean that the operations domain can't be expected to know how to update QoH numbers in the sales domain (and vice versa, of course).

Domains, therefore, might well be querying one database (their own) while sending commands to another database (everyone sends QoH updates to the sales domain, for example). While DDD provides a strategy for segmenting domains with different business requirements, CQRS provides one of the tactics for managing updates among those domains (for a more in-depth discussion of the query side of CQRS, see Esposito's March 2016 column, "The Query Stack of a CQRS Architecture," at [bit.ly/1WzjvPi](http://bit.ly/1WzjvPi)).

### Handling Commands and Events

Of course, you don't want to make the applications in these domains more complicated by having to deal with the diversity of domains that must be notified for each transaction. Rather than keep track of all of the domains that must be updated, each application will send transactions to a utility that's responsible for notifying the various domains (typically called a "command bus"). As new domains are defined (or existing domains change their demands), only the command bus within the domain that originates the transaction needs to be updated to reflect the new notifications that are required.

These transactions can be divided into categories: commands and events. The distinction between the two is more conceptual than technical. Effectively, both commands and events are messages that wrap up the key information about a transaction. For our

inventory transactions that would be the Id for the SKU, the net change to the inventory level, and the additional data required by the transaction (when goods are received that additional data might be the vendor number and the invoice number; during stock taking the additional data might be the Id of the employee who actually counted the SKUs). These messages could be encoded as POJO objects or as XML/JSON documents (or both, depending on how the data is sent between domains).

For me, the definition of a command is that it's something directed to a single receiver in order to carry out a task. A command is usually a task that needs to be performed immediately and, obviously, is sent before the task is performed. A command can also be expected to return a success/failure response that the application can use to inform the user whether everything has worked (and, potentially, cause the application to perform a query to retrieve the data that shows what results were achieved). Most updates within the domain that originated the transaction are probably handled with commands.

Events, on the other hand, occur after the task is performed, can be processed by multiple receivers and, usually, aren't required to be processed immediately. Events aren't expected to return a result, at least not immediately. If something goes wrong with an event, the application will typically find out about it through some deferred return message ("We're sorry, it turns out we can't process your order because your credit card was declined"). Most, but not all, updates outside of the domain that originated the transaction are probably handled with events.

And, like most conceptual distinctions, this is probably a continuum; some messages are "obviously" commands, some messages are "obviously" events and there are some about which reasonable people could disagree.

A single transaction in one domain might generate some combination of commands and events. Consider new SKUs showing up at the receiving dock. Once the SKUs are properly received, the bus for that domain would send a command to the sales system to have the QoH for that SKU increased immediately; the bus would also post an event so that the accounting and operations systems can be notified that "something has happened" and should be taken into account at month's end. Looking at the messages involved, it might be difficult to determine which one is the event and which one is the command—except, perhaps, by looking at the name of the message; events tend to have names in the past tense (GoodsReceived) while commands tend to have imperative names (IncreaseInventory).

The bus might send the command to the sales system by calling a RESTful service in that domain for immediate execution; the event might be written to some message queue to be processed by other domains at their convenience (I've discussed some of the options in an article I wrote for *VisualStudioMagazine.com*, "Simplifying Applications by Implementing Eventual Consistency with Domain Events," at [bit.ly/1qn1wwV](http://bit.ly/1qn1wwV)).

Of course, even with the command sent to the Web service, who knows what happens behind that Web service? In order to handle large numbers of simultaneous requests, the Web service for the domain might just write the command message to a queue and return a "Thanks, got it" response, keeping response time short and



# Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn  
magazine

[msdn.microsoft.com/flashnewsletter](https://msdn.microsoft.com/flashnewsletter)

improving scalability. In addition to improving scalability, writing commands to a queue lets the domain recover from what might otherwise be catastrophic problems. If the database or network is down, for example, the sales system can wait patiently for service to be restored and then process any commands sitting on its queue. So, even commands can end up on queues.

As I said, the distinction between commands and events is conceptual, not technical.

## Processing Commands and Events

Thanks to CQRS, applications can now be working with some combination of two databases: one for queries (probably local to the domain) and other databases that are targets for commands and events. For example, the sales system will be working with a data store that includes the NoSQL database that holds QoH data; the operations and accounting applications might be working with a data store organized around the history of the events/commands.

The difference between the two systems is that the sales system needs a snapshot of the current state of the inventory levels to meet response time demands; the operations and accounting domains need a history of what happened to each SKU to support analysis. The operations and accounting domains can work by using a different tactic: event sourcing. With event sourcing, domain logic rolls through the audit log of the events they've been notified about to provide a final answer (for the accounting system that might be, "Based on the history of posted transactions, the current value of your inventory is X dollars").

Thanks to CQRS, applications can now be working with some combination of two databases: one for queries (probably local to the domain) and other databases that are targets for commands and events.

There are advantages and disadvantages to event sourcing. With event sourcing it's always possible to recreate a snapshot of the current state of the data by reprocessing the list of transactions; accounting appreciates that feature as it adds adjustments to the list of events. With event sourcing, it's also possible to describe the future by processing potential events (expected deliveries and sales); operations appreciate that feature when planning.

As the list of events increases, so does response time, however. Accounting can roll forward from its "last known good state" (probably the numbers from the last month's-end closing) and generate a snapshot that represents the month's-end numbers. That

snapshot is saved as the current "last known good state" and published as month's-end reports. Operations can roll forward from today to some indeterminate point in the future and, probably, never generate a snapshot; they'd recreate the future each time it was requested. Given the response time expectations for these domains, those are probably reasonable scenarios.

To determine the current QoH for the sales system using event sourcing, however, the sales system would have to roll forward through all transactions since the last inventory count. Because inventory counts are labor intensive, those counts don't occur very often. As a result, processing all of those events since the last count would create unacceptable response times for the sales system. Instead, the sales system keeps its constantly updated QoH numbers in memory.

## Wrapping Up

While querying requires various levels of support in database (and, as a result, a variety of indexes and foreign/primary keys), updates do not. Virtually all updates are driven by the Ids of the entities involved. The list of inventory SKU with the QoH numbers, for example, is driven entirely by the SKU Id. This can dramatically simplify the data model for the command side of a CQRS system. The ability for Entity Framework to generate a collection of `SalesOrderItems` for a `SalesOrder` is irrelevant if the command/event message simply includes the Ids for all of the `SalesOrderItems` that were changed in a transaction.

The impact on locking in the database as a result of this design is interesting. Updates to the QoH and reserved quantities in the sales system consist of changing one or both of the integer values; locking should be minimal. Locking in some of the other systems can disappear if those systems are event sourced; the transaction always inserts some transaction into an event table so there are no updates.

Effectively, then, the business has multiple, independent processors updating data within their domains, processing commands and events. Without locking, it's possible that this can create conflicts. For example, a command to purchase two items might appear at the same time as an event that reduces the QoH to zero (someone did an inventory count and noticed that there was nothing on the shelf). Interestingly, a queue-based, event-sourcing approach might resolve this problem; the QoH update processor in the sales system could work on an event-sourcing basis, rolling through all the recently received commands in a queue (within some limit) and updating QoH with the total of their results. Commands that show up *simultaneously* would be summarized into a single update. Alternatively, it might simply be necessary to recognize that, on occasion, the business is allowed to cancel an order just like a user can.

CQRS is a powerful tool. It has the biggest payoff, however, when applied to shared data stores, collaborative processes and within the strategy provided by DDD. ■

---

**PETER VOGEL** is a system architect and principal in PH&V Information Services. PH&V provides full-stack consulting from UX design through object modeling to database design. You can contact him at [peter.vogel@phvis.com](mailto:peter.vogel@phvis.com).

---

**THANKS** to the following Microsoft technical experts for reviewing this article: Dino Esposito and Julie Lerman



# JOIN US on the CAMPAIGN TRAIL in 2016!



**AUGUST 8 - 12**

MICROSOFT HQ, REDMOND, WA

**[vslive.com/redmond](http://vslive.com/redmond)**

See pages 66–69 for more info



**SEPTEMBER 26 - 29**

HYATT ORANGE COUNTY, CA –  
A DISNEYLAND® GOOD  
NEIGHBOR HOTEL

**[vslive.com/anaheim](http://vslive.com/anaheim)**

See pages 44–45 for more info



**OCTOBER 3 - 6**

RENAISSANCE, WASHINGTON, D.C.

**[vslive.com/dc](http://vslive.com/dc)**

See pages 46–47 for more info



**PART OF LIVE! 360**

**DECEMBER 5 - 9**

LOEWS ROYAL PACIFIC  
ORLANDO, FL

**[vslive.com/orlando](http://vslive.com/orlando)**

See pages 56–57 for more info



CONNECT WITH VISUAL STUDIO LIVE!



[twitter.com/vslive](https://twitter.com/vslive) – @VSLive



[facebook.com](https://www.facebook.com/vslive) – Search “VSLive”



[linkedin.com](https://www.linkedin.com/groups?gid=11111111) – Join the  
“Visual Studio Live” group!

**VSLIVE.COM**



CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

# CODE

## Anaheim

### FOR A BETTER TOMORROW



**SEPTEMBER 26-29, 2016**

**HYATT REGENCY  
DISNEYLAND®**

**GOOD NEIGHBOR HOTEL**

**VISUAL STUDIO LIVE!** (VSLive!™) is blazing new trails on the Campaign for Code: For the first time ever, we are headed to Anaheim, CA, to bring our unique brand of unbiased developer training to Southern California. With our host hotel situated just down the street from Disneyland®, developers, software architects, engineers, designers and more can code by day, and visit the Magic Kingdom by night. From Sept. 26-29, explore how the code you write today will create a better tomorrow—not only for your company, but your career, as well!

**California code with us:  
register to join us today!**

**REGISTER BY JULY 20  
& SAVE \$300!**

Scan the QR code to register or  
for more event details.



**USE PROMO CODE VSLANS**

SUPPORTED BY



Visual Studio  
MAGAZINE



PRODUCED BY



# ANAHEIM AGENDA AT-A-GLANCE

ALM / DevOps	ASP.NET	Cloud Computing	Database and Analytics	JavaScript / HTML5 Client	Mobile Client	Software Practices	Visual Studio / .NET Framework	Windows Client
Visual Studio Live! Pre-Conference Workshops: Monday, September 26, 2016 <i>(Separate entry fee required)</i>								
START TIME	END TIME	Pre-Conference Workshop Registration - Coffee and Morning Pastries						
7:30 AM	9:00 AM							
9:00 AM	6:00 PM	<b>M01</b> Workshop: Building for the Internet of Things: Hardware, Sensors & the Cloud - Nick Landry	<b>M02</b> Workshop: Developer Dive into SQL Server 2016 - Leonard Lobel	<b>M03</b> Workshop: Distributed Cross-Platform Application Architecture - Rockford Lhotka & Jason Bock			<b>M04</b> Workshop: Creating Awesome 2D & 3D Games and Experiences with Unity - Adam Tuliper	
6:45 PM	9:00 PM	Dine-A-Round						
Visual Studio Live! Day 1: Tuesday, September 27, 2016								
START TIME	END TIME	Registration - Coffee and Morning Pastries						
7:00 AM	8:00 AM							
8:00 AM	9:00 AM	Keynote: To Be Announced						
9:15 AM	10:30 AM	<b>T01</b> Angular 2 101 - Deborah Kurata	<b>T02</b> Developing Universal Windows Platform (UWP) Applications - Scott Golightly	<b>T03</b> Developer Productivity in Visual Studio 2015 - Robert Green			🔴 <b>T04</b> This session is sequestered, details will be released soon	
10:45 AM	12:00 PM	<b>T05</b> ASP.NET MVC6—What You Need to Know - Philip Japikse	<b>T06</b> Building Connected and Disconnected Mobile Applications - James Montemagno	<b>T07</b> Professional Scrum Development Using Visual Studio 2015 - Richard Hundhausen			🔴 <b>T08</b> This session is sequestered, details will be released soon	
12:00 PM	1:30 PM	Lunch - Visit Exhibitors						
1:30 PM	2:45 PM	<b>T09</b> Angular 2 Forms and Validation - Deborah Kurata	<b>T10</b> New SQL Server 2016 Security Features for Developers - Leonard Lobel	<b>T11</b> Use Visual Studio to Scale Agile in Your Enterprise - Richard Hundhausen			<b>T12</b> Windows for Makers: Raspberry Pi, Arduino & IoT - Nick Landry	
3:00 PM	4:15 PM	<b>T13</b> Introduction to ASP.NET Core 1.0 - Scott Golightly	<b>T14</b> Cross-platform Mobile Development for the C# Developer - James Montemagno	🔴 <b>T15</b> This session is sequestered, details will be released soon			<b>T16</b> Cloud Enable .NET Client LOB Applications - Robert Green	
4:15 PM	5:30 PM	Welcome Reception						
Visual Studio Live! Day 2: Wednesday, September 28, 2016								
START TIME	END TIME	Registration - Coffee and Morning Pastries						
7:00 AM	8:00 AM							
8:00 AM	9:15 AM	<b>W01</b> JavaScript Patterns for the C# Developer - Ben Hoelting	<b>W02</b> Implementing the Mvvm Pattern in Your Xamarin Apps - Kevin Ford	<b>W03</b> VSTS and Azure: Cloud DevOps 101 - Mike Benkovich			<b>W04</b> Introduction to Next Generation of Azure PaaS—Service Fabric and Containers - Vishwas Lele	
9:30 AM	10:45 AM	<b>W05</b> Busy Developer's Guide to TypeScript - Ted Neward	🔴 <b>W06</b> This session is sequestered, details will be released soon	<b>W07</b> User Story Mapping - Philip Japikse			<b>W08</b> Cloud Oriented Programming - Vishwas Lele	
11:00 AM	12:00 PM	<b>GENERAL SESSION: To Be Announced</b> - James Montemagno, Developer Evangelist, Xamarin Team, Microsoft						
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors						
1:30 PM	2:45 PM	<b>W09</b> Getting Started with ASP.NET Core 1 Web API and Entity Framework Core 1 - Philip Japikse	<b>W10</b> Using Visual Studio Tools for Apache Cordova to Create MultiPlatform Applications - Kevin Ford	<b>W11</b> Exploring C# 7 New Features - Adam Tuliper			<b>W12</b> Lock the Doors, Secure the Valuables, and Set the Alarm - Eric D. Boyd	
3:00 PM	4:15 PM	<b>W13</b> Angular 2 and ASP.NET Core—A Perfect Match! - Dan Wahlin	<b>W14</b> Busy .NET Developer's Guide to Swift - Ted Neward	<b>W15</b> I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - Adam Tuliper			<b>W16</b> App Services: The New PaaS - Mike Benkovich	
4:30 PM	5:45 PM	<b>W17</b> Increase Website Performance and Search with Lucene.Net Indexing - Ben Hoelting	<b>W18</b> Reusing Business Logic in Cross-Platform .NET - Rockford Lhotka	<b>W19</b> Database Development with SQL Server Data Tools - Leonard Lobel			<b>W20</b> Breaking Down Walls with Modern Identity - Eric D. Boyd	
7:00 PM	9:00 PM	Evening Event						
Visual Studio Live! Day 3: Thursday, September 29, 2016								
START TIME	END TIME	Registration - Coffee and Morning Pastries						
7:00 AM	8:00 AM							
8:00 AM	9:15 AM	<b>TH01</b> Building Data-Centric Apps with Angular 2 and Breeze - Brian Noyes	<b>TH02</b> Building Business Apps on the Universal Windows Platform - Billy Hollis	<b>TH03</b> Using Your SQL Skills to Become a Big Data Developer - Omid Afnan			<b>TH04</b> Docker: Creating the Ultimate Development Environment - Dan Wahlin	
9:30 AM	10:45 AM	<b>TH05</b> Getting Started with Aurelia - Brian Noyes	<b>TH06</b> HoloLens - Billy Hollis	<b>TH07</b> Data Analytics with Azure Data Lake: U-SQL In Depth - Omid Afnan			<b>H08</b> DevOps for Developers - Brian Randell	
11:00 AM	12:15 PM	<b>TH09</b> Pretty, Yet Powerful. How Data Visualization Transforms the Way We Comprehend Information - Walt Ritscher	<b>TH10</b> Dependencies Demystified - Jason Bock	<b>TH11</b> Predicting the Future Using Azure Machine Learning - Eric D. Boyd			<b>TH12</b> PowerShell for Developers - Brian Randell	
12:15 PM	1:30 PM	Lunch						
1:30 PM	2:45 PM	<b>TH13</b> Building Rich UI with ASP.NET MVC and Bootstrap - Walt Ritscher	<b>TH14</b> What's the Problem?!?: How to Resolve Conflict - Robert Bogue	<b>TH15</b> Exploratory Data Analysis with R Tools for Visual Studio - Matthew Renze			<b>TH16</b> Developing Awesome 3D Apps with Unity and C# - Adam Tuliper	
3:00 PM	4:15 PM	<b>TH17</b> Assembling the Web - A Tour of WebAssembly - Jason Bock	<b>TH18</b> Hack Proof: Software Design for a Hostile Internet - Robert Bogue	<b>TH19</b> Data Visualization with R Tools for Visual Studio - Matthew Renze			<b>TH20</b> Technical Debt - Fight It with Science and Rigor - Brian Randell	

Speakers and sessions subject to change



**DETAILS COMING SOON!** These sessions have been sequestered by our conference chairs. Be sure to check [vslive.com/anaheim](http://vslive.com/anaheim) for session updates!

CONNECT WITH VISUAL STUDIO LIVE!



twitter.com/vslive - @VSLive



facebook.com - Search "VSLive"



linkedin.com - Join the "Visual Studio Live" group!

**VSLIVE.COM/ANAHEIM**



# VOTE "YES" FOR BETTER

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

# CODE

## Washington, D.C.

**OCT. 3-6, 2016**

**RENAISSANCE  
WASHINGTON, D.C.**



**VISUAL STUDIO LIVE!** (VSLive!™) is on a Campaign for Code in 2016, in support of developer education. It's only fitting that we return with our unique brand of practical, unbiased, Developer training to the nation's capital this year. From Oct. 3-6, we're offering four days of sessions, workshops and networking events to developers, software architects, engineers and designers—all designed to help you vote "yes" for better code and write winning applications across all platforms.

**Do your developer duty:  
register to join us today!**

**REGISTER BY AUGUST 3  
& SAVE \$300!**

Scan the QR code to register or  
for more event details.

**USE PROMO CODE VSLDCS**



SUPPORTED BY



Visual Studio  
MAGAZINE



PRODUCED BY





# WASHINGTON, D.C. AGENDA AT-A-GLANCE

ALM / DevOps	Cloud Computing	Database and Analytics	Mobile Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server	Windows Client
Visual Studio Live! Pre-Conference Workshops: Monday, October 3, 2016 <i>(Separate entry fee required)</i>								
START TIME	END TIME	Pre-Conference Workshop Registration • Coffee and Morning Pastries						
7:30 AM	9:00 AM							
9:00 AM	6:00 PM	<b>M01</b> Workshop: DevOps in a Day - Brian Randell			<b>M02</b> Workshop: SQL Server for Developers - Andrew Brust and Leonard Lobel		<b>M03</b> Workshop: Distributed Cross-Platform Application Architecture - Rockford Lhotka & Jason Bock	
6:45 PM	9:00 PM	Dine-A-Round						
Visual Studio Live! Day 1: Tuesday, October 4, 2016								
START TIME	END TIME	Registration - Coffee and Morning Pastries						
7:00 AM	8:00 AM							
8:00 AM	9:00 AM	Keynote: To Be Announced						
9:15 AM	10:30 AM	<b>T01</b> AngularJS2 in 75 Minutes - Sahil Malik	★ <b>T02</b> This session is sequestered, details will be released soon		<b>T03</b> Developer Productivity in Visual Studio 2015 - Robert Green		<b>T04</b> Cloud Oriented Programming - Vishwas Lele	
10:45 AM	12:00 PM	<b>T05</b> ASP.NET MVC6—What You Need to Know - Philip Japikse	<b>T06</b> What's New in SQL Server 2016 - Leonard Lobel		<b>T07</b> HoloLens - Brian Randell		★ <b>T08</b> This session is sequestered, details will be released soon	
12:00 PM	1:30 PM	Lunch - Visit Exhibitors						
1:30 PM	2:45 PM	<b>T09</b> AngularJS2 with Web, Mobile (Cordova), and Desktop (electron) - Sahil Malik	<b>T10</b> Database Development with SQL Server Data Tools - Leonard Lobel		★ <b>T11</b> This session is sequestered, details will be released soon		<b>T12</b> Windows for Makers: Raspberry Pi, Arduino & IoT - Nick Landry	
3:00 PM	4:15 PM	<b>T13</b> Getting Started with ASP.NET Core 1 Web API and Entity Framework Core 1 - Philip Japikse	<b>T14</b> Cross-platform Mobile Development for the C# Developer - James Montemagno		<b>T15</b> Exploring C# 7 New Features - Adam Tuliper		<b>T16</b> Cloud Enable .NET Client LOB Applications - Robert Green	
4:15 PM	5:30 PM	Welcome Reception						
Visual Studio Live! Day 2: Wednesday, October 5, 2016								
START TIME	END TIME	Registration - Coffee and Morning Pastries						
7:00 AM	8:00 AM							
8:00 AM	9:15 AM	<b>W01</b> JavaScript Patterns for the C# Developer - Ben Hoelting	<b>W02</b> Implementing the Mvvm Pattern in Your Xamarin Apps - Kevin Ford		<b>W03</b> Developing Awesome 3D Apps with Unity and C# - Adam Tuliper		<b>W04</b> Introduction to Next Generation of Azure PaaS—Service Fabric and Containers - Vishwas Lele	
9:30 AM	10:45 AM	<b>W05</b> Busy Developer's Guide to TypeScript - Ted Neward	<b>W06</b> Building Connected and Disconnected Mobile Applications - James Montemagno		<b>W07</b> Unleash the New .NET: Open Source and Running on OS X, Linux & Windows - Nick Landry		★ <b>W08</b> This session is sequestered, details will be released soon	
11:00 AM	12:00 PM	<b>GENERAL SESSION: More Personal Computing through Emerging Experiences</b> - Tim Huckaby, Founder / Chairman - Interknowlogy & Actus Interactive Software						
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors						
1:30 PM	2:45 PM	<b>W09</b> WCF & Web API: Can We All Just Get Along?!? - Miguel Castro	<b>W10</b> Using Visual Studio Tools for Apache Cordova to Create MultiPlatform Applications - Kevin Ford		<b>W11</b> Software Engineering in an Agile Environment - David Corbin		<b>W12</b> Lock the Doors, Secure the Valuables, and Set the Alarm - Eric D. Boyd	
3:00 PM	4:15 PM	<b>W13</b> Richer MVC Sites with Knockout JS - Miguel Castro	<b>W14</b> Busy .NET Developer's Guide to Swift - Ted Neward		<b>W15</b> Technical Debt—Fight It with Science and Rigor - Brian Randell		<b>W16</b> Breaking Down Walls with Modern Identity - Eric D. Boyd	
4:30 PM	5:45 PM	<b>W17</b> Increase Website Performance and Search with Lucene.Net Indexing - Ben Hoelting	<b>W18</b> Reusing Business Logic in Cross-Platform .NET - Rockford Lhotka		<b>W19</b> Implementing Data Protection and Security in SQL Server 2016 - Steve Jones		<b>W20</b> Build Real-Time Websites and Apps with SignalR and Azure - Rachel Appel	
6:45 PM	10:30 PM	Visual Studio Live! Monuments by Moonlight Tour						
Visual Studio Live! Day 3: Thursday, October 6, 2016								
START TIME	END TIME	Registration - Coffee and Morning Pastries						
7:00 AM	8:00 AM							
8:00 AM	9:15 AM	<b>TH01</b> Building Data-Centric Apps with Angular 2 and Breeze - Brian Noyes	<b>TH02</b> Moving from WPF to Windows 10 and Universal Apps - Billy Hollis		<b>TH03</b> Test Drive Automated GUI Testing with WebDriver - Rachel Appel		<b>TH04</b> Big Data and Hadoop with Azure HDInsight - Andrew Brust	
9:30 AM	10:45 AM	<b>TH05</b> Getting Started with Aurelia - Brian Noyes	<b>TH06</b> Building Business Apps on the Universal Windows Platform - Billy Hollis		<b>TH07</b> Exposing an Extensibility API for your Applications - Miguel Castro		<b>TH08</b> Power BI 2.0: Analytics in the Cloud and in Excel - Andrew Brust	
11:00 AM	12:15 PM	<b>TH09</b> Dependencies Demystified - Jason Bock	<b>TH10</b> Pretty, Yet Powerful. How Data Visualization Transforms the Way We Comprehend Information - Walt Ritscher		<b>TH11</b> Bringing DevOps to the Database - Steve Jones		<b>TH12</b> Predicting the Future Using Azure Machine Learning - Eric D. Boyd	
12:15 PM	1:30 PM	Lunch						
1:30 PM	2:45 PM	<b>TH13</b> Building Rich UI with ASP.NET MVC and Bootstrap - Walt Ritscher	<b>TH14</b> What's the Problem?!?: How to Resolve Conflict - Robert Bogue		<b>TH15</b> Build Distributed Business Apps Using CSLA .NET - Rockford Lhotka		<b>TH16</b> Exploratory Data Analysis with R Tools for Visual Studio - Matthew Renze	
3:00 PM	4:15 PM	<b>TH17</b> Assembling the Web—A Tour of WebAssembly - Jason Bock	<b>TH18</b> Hack Proof: Software Design for a Hostile Internet - Robert Bogue		<b>TH19</b> Bootstrapping Automated Testing for Existing Software Systems - David Corbin		<b>TH20</b> Data Visualization with R Tools for Visual Studio - Matthew Renze	

Speakers and sessions subject to change

★ DETAILS COMING SOON! These sessions have been sequestered by our conference chairs. Be sure to check [vslive.com/dc](http://vslive.com/dc) for session updates!

CONNECT WITH VISUAL STUDIO LIVE!



[twitter.com/vslive](https://twitter.com/vslive) – @VSLive



[facebook.com](https://facebook.com/vslive) – Search “VSLive”



[linkedin.com](https://linkedin.com/vslive) – Join the “Visual Studio Live” group!

**VSLIVE.COM/DC**

# Applying AI to a Multi-Agent Mini-Basketball Game

Arnaldo Pérez Castaño

**Artificial intelligence (AI)** is now the most interesting, researched field in computer science and many of its sub-fields have great impact on our daily lives. You can see AI applied almost everywhere and in the most unexpected scenarios—when an insurance company applies data mining algorithms to find relations between individuals (making sure a client isn't attempting a scam), when you get a list of suggested friends on Facebook, when you play a video game and so on. Its applications are enormous and include areas such as economics, commerce, astronomy, medicine and military.

The desire to automate diverse human tasks and to create an artificial conscience has led, over time, to the formation of this popular

science known as AI. Its definition, simple as it can sound, regularly produces doubts because AI represents probably the broadest field in computer science. From an expert's point of view, AI is the automation of activities that we associate with human thinking and activities such as decision making, problem solving, learning and so on ("Artificial Intelligence: Can Computers Think?" Richard E. Bellman, 1978). The previous definition states how AI relates to automating human activities. Thus, the question arises: Why do you need to create AI to automate human tasks? Is it really necessary? The answer is yes.

Humans are the most sophisticated, elegant and complicated bio-machines known to mankind. The complex and fascinating functioning of our organisms, combined with the amazing capabilities of our brain, make us almost perfect machines. But, if humans are so perfect, then why create e-steel machines and AIs?

First, humans create electronic, steel machines because even though our organism functions incredibly well, it's very fragile; therefore, it can't stand extremely high or low temperatures, it requires oxygen to work properly, it can be easily damaged by natural objects, it lacks the strength that steel possesses and so on. An electronic, steel machine bypasses all of these tribulations.

Second, the human brain, although capable of experiencing emotions and making some really complicated reasoning, turns slow when compared to computers under a simple and primitive criteria: the calculation. A machine brain (computer), unlike a human brain, is capable of executing millions of calculations per

## This article discusses:

- AI, agents and multi-agents system in a basketball game environment
- Implementation of Finite State Machines and Behavior Trees in such environment
- Development of a multithread C# application for simulating a basketball game

## Technologies discussed:

C#, Windows Forms, Microsoft .NET Framework, AI Applications

## Code download available at:

[msdn.com/magazine/0716magcode](http://msdn.com/magazine/0716magcode)

second. Operations such as searching and sorting in big domains are accomplished much faster by computers than humans. Hence, AIs are necessary to help us make our life more efficient and to save time. That's the case when you use a calculator; the human brain is usually incapable of providing a fast calculation for the square root of a big number, for example. Thus, you use a calculator to obtain that result almost instantaneously; a calculator is basically a robot whose task is calculating.

The desire to automate diverse human tasks and to create an artificial conscience has lead, over time, to the formation of this popular science known as AI.

In the next sections, I'll describesome of the traditional entities, systems related to AI and their interactions. I'll show how to develop an AI for a multi-agent basketball game,where it's possible to have two basketball teams (Bulls and Knicks) playing against each other.

## Agents

The '90s brought the concept of agents in computer science and the term is now as modern and fashionable as object-oriented was in the '80s or AI in the '70s. An agent is an entity capable of sensing its environment and acting upon it ("Artificial Intelligence: A Modern Approach," Stuart Russell and Peter Norvig, 1997). The main difference between an agent and an ordinary program is that the first must be autonomous; that is, it must operate without direct intervention of humans or others. An additional difference is that the agent performs specific tasks on behalf of someone else (usually known as the user or programmer), hence the word agent, as it suggests, indicates someone who just acts on behalf of others.

A rational agent is one that acts so as to achieve the best outcome or, when there's uncertainty, the best expected outcome ("Artificial Intelligence: A Modern Approach," Stuart Russell and Peter Norvig, 1997). Rationality in this sense refers to making correct inferences and selecting (whenever possible) the action that after a logical conclusion will lead to achieving the desired goal. Humans are rational agents; they sense their environment through their eyes,

ears, tongue, and nose and eventually act using their legs, hands and so on, generally after applying some logical reasoning and selecting the actions that'll lead them to their desired goal.

A percept refers to the agent's perceptual inputs at any given moment. The percept sequence represents the complete sequence of percepts the agent has sensed or perceived during his lifetime. The agent function represents the agent behavior through a mapping from any percept sequence to an action. This function is merely an abstract description; the agent program is the actual implementation of this abstraction. **Figure 1** shows the partial agent function for a basketball player. The column labeled Percept Sequence contains the player's states and the column labeled Action contains the action to be executed after the corresponding percept sequence.

You could partition the world of agents, according to their architecture, in the following categories:

- A reactive agent is capable of maintaining an ongoing interaction with the environment and responding in a timely fashion to changes that occur in it. The term is now widely used to mean a system that includes no symbolic representation or reasoning; such an agent doesn't reflect on the long-term effects of its action, and doesn't consider the coordination of activity with other agents. Thus, a reactive agent will always respond in a timely fashion to external stimulus and is event-driven. This can be implemented by simple if-then rules. The agent's goals are only implicitly represented by the rules and it's hard to ensure the desired behavior. Each and every situation must be considered in advance; hence, reactive systems in complex environments usually contain hundreds of rules. Reactive agents have no internal model of the world, therefore are incapable of reasoning about it in any abstract way. To reiterate, the agent simply receives inputs and reacts to these through simple rules.
- A pro-active agent is capable of taking the initiative; not driven solely by events, but capable of generating goals and acting rationally to achieve them. Some see it as a goal-driven reactive agent.
- A deliberative agent symbolically represents knowledge and makes use of mental notions such as beliefs, intentions, desires, choices and so on. It tries to model human reasoning, distributed activities and behavior through logical representations. It's usually implemented by means of believe, desire, intention (BDI) architecture. It can reason about the past and plan into the future; planning is essential in this type of architecture.
- A hybrid agent is one that mixes some of all the different architectures.

Another alternative to partition the world of agents is by dividing it into learning and non-learning agents. A learning agent is one that requires some training to perform well, adapts its current behavior based on previous experiences and evolves over time. A non-learning agent is one that doesn't evolve or relates to past experiences and is hardcoded and independent of its programming.

Because the basketball game environment is discreet, finite and defined by a finite set of rules, I'll propose a non-learning agent in this article.

**Figure 1 Partial Agent Function for a Basketball Player**

Percept Sequence	Action
(close to basket, unguarded)	Shoot
(close to opponent, opponent shoots)	Block
(opponent loses ball)	Steal
(guarded, teammate unguarded)	Pass
(teammate close to basket, teammate unguarded, teammate good slammer, eye connection)	Alley-oop Pass

## Multi-Agent System

When an agent coexists in an environment with other agents—perhaps collaborating or competing with them—it's considered a multi-agent system (MAS). In MAS environments, each agent has its own perspective of the world and no knowledge of the internal states or the manner in which other agents see the environment. Thus, MAS represents a type of distributed system with the following features:

- Agents have incomplete information about the system or insufficient capabilities for solving a task autonomously.
- The system exhibits no global control.
- Data is decentralized.

A coalition is any subset of agents in the environment. For the basketball game there are two coalitions—team A and team B—whose intersection is empty and who both have the same cardinality, which is greater than zero.

A strategy is a function that receives the current state of the environment and outputs the action to be executed by a coalition. The strategy for team A usually depends on the actions executed by each agent in team B at the current moment.

In MAS, interaction occurs through means of communication and there exists various forms by which agents can communicate; plain signals matched to fixed interpretations are probably the naive form of communication among agents. A blackboard structure is a communication form that consists of a shared resource divided into different areas of the environment where agents can read or write any significant information for their actions. Message passing between agents is another form of communication. In this form, agents exchange messages with a given syntax and protocol but lack semantics; thus, the receiving agent must deduce the intention of the message. Finally, speech act theory greatly impeded by American philosopher John R. Searle in 1969 (“Speech Acts: An Essay in the Philosophy of Language”) and Canadian logician Daniel Vanderveken in 1994 (“Foundations of Speech Act Theory”) overcomes the disadvantage of message passing by taking the interaction among agents in two levels—first, the informational content of the message and, second, the intention of the message. This approach marks a difference between the locution act (words, sentences), the locution intention (inform, request, order and so on) and the desired result of the locution (insult, convince and so on).

Coordination is essential in MAS because it provides coherency to the system behavior and contributes to achieving team or coalition goals. It implies taking into account actions from other agents for planning and executing the ones corresponding to individual or multiple agents. In many cases, coordination also implies cooperation or competition; both exist in a basketball game.

Cooperation is necessary as a result of complementary skills and the interdependency present among agent actions and the inevitability of

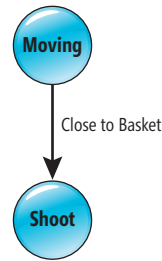


Figure 2 A Simple Finite State Machine Model

satisfying some success criteria. In a cooperative model, agents are collectively motivated or collectively interested; hence, they work to accomplish a common goal. Another possible model is that in which the agents are self-motivated or self-interested agents because each agent has its own goals and might enter into competition with the other agents in the system to achieve these goals. In this sense, competition might refer to accomplishing or distributing certain tasks. In such a model, agents need to coordinate their actions with other agents to guarantee a coherent behavior. During the coordination process, both in a cooperative or a competitive environment, conflicts might appear and these are solved

by means of negotiation. Negotiation might be seen as the process of identifying interactions based on communication and reasoning regarding the state and intentions of other agents.

In the next section, I'll describe a novel and interesting data structure that's present in many of today's video games: the behavior tree.

## Finite State Machines and Behavior Trees

Finite state machines (FSMs) have been the traditional approach for modeling decision making, action selection and execution of agent behavior in games. An FSM represents a mathematical model conformed by a finite set of states and a transition function. FSMs provide a simple, intuitive mechanism for reactive behavior, processing and reacting to continuous streams of events or inputs.

Figure 2 illustrates a simple FSM modeling the behavior of an agent once it reaches a distance to the basket considered as “close.” When that occurs, it moves to the Shoot behavior. In this scenario, states correspond to behaviors and transitions to events. The main disadvantage of an FSM comes when the need to extend its functionality or implement a more complex behavior appears. In such cases, the number of state transitions could increase exponentially making the FSM extremely hard to understand and process.

Behavior trees (BTs) provide a simple, scalable, modular solution to represent complex AI behaviors offering an easy-to-maintain and set-up logic. Their use in the game industry has increased greatly in the last few years where titles such as “Halo3,” “Spore,” “BioShock” and “SWAT 4” included BTs as behavior-modeling tools. BTs are goal-oriented and each tree is related to a distinct, high-level goal. BTs can be linked together, allowing the implementation of complex behaviors by first defining smaller, sub-behaviors.

Each node in a BT is either a primitive construct or a composite construct. The first type forms the leaves of the tree; the latter represents a manner to describe relationships between child nodes.

There are two types of primitive constructs. Actions embody the execution of a method related to the agent (move, shoot and others). Conditions query the state of the environment (is opponent reachable, is move feasible, is close to basket and others).

Figure 3 shows a BT with two children nodes—a condition C and an action A.

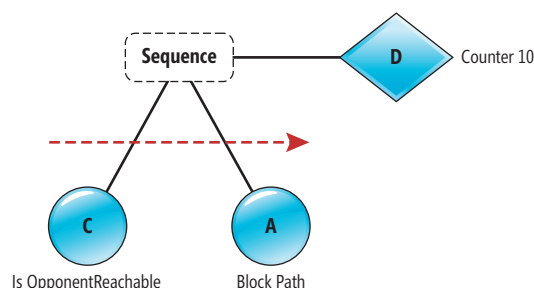


Figure 3 A Behavior Tree with Two Children Nodes



Whitepaper Available  
Four Ways to Optimize ASP.NET Performance

# Extreme Performance Linear Scalability

Cache data, reduce expensive database trips, and scale your apps to extreme transaction processing (XTP) with NCache.

## In-Memory Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- Entity Framework & NHibernate Second Level Cache

## ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider

## Full Integration with Microsoft Visual Studio

- NuGet Package for NCache SDK
- Microsoft Certified for Windows Server 2012 R2



[sales@alachisoft.com](mailto:sales@alachisoft.com)

US: +1 (925) 236 3830

**FREE Download**  
[www.alachisoft.com](http://www.alachisoft.com)

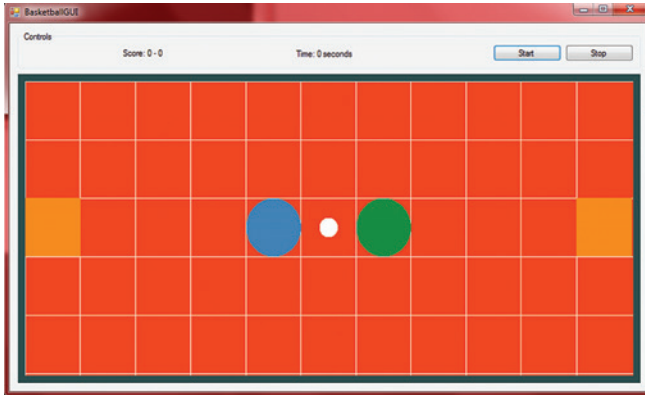


Figure 4 A Basketball Game with AI Implementation

There are four types of composite constructs. Selectors pick one of its children to be executed. This selection can be done randomly or using some priority. The value of the selector depends on whether the child node executed was successful (return true) or not.

Sequences impose an ordering in the execution of children nodes. In **Figure 3**, the red-dotted line indicates the ordering of execution in the sequence tree. For this type of node to be successful, each and every one of its children nodes must be successful, as well.

Finally, Decorators, which are inspired in the programming design pattern, provide a way to simplify the programming process and extend behavior by adding functionality to a node. A Timer Decorator could be a kind of decorator that, as the name suggests, executes its children nodes after certain time intervals. A Counter Decorator, on the other hand, could execute a child node or behavior several times. In **Figure 3**, the D counter decorator will execute the node Sequence 10 times, as defined.

In the next section, I'll describe the simplified version of the basketball game considered in this article and the AI proposed for such a game.

## A Basketball Game and AI Implementation

Because a basketball game is a big, complicated game that would require a complex FSM or a really big BT, only a simplified version of it that includes some basic strategies for offensive and defensive plays will be considered in this article. In this game, there'll be two players (A1->blue, B1->green), each with properties Speed and Shooting Accuracy. The first defines how fast or frequent the player reacts to the environment and the latter defines the probability that a player will make a shot once attempted. Initially, players start in the center as shown in **Figure 4**. The score is 0-0 and the time is 0 seconds. If a player hasn't scored after 15 seconds, a shot-clock violation is triggered and that player loses possession of the ball. The GUI of the game consists of a Windows Forms application, which you can later consult in

Figure 5 Court Class

```
public class Court
{
    public int Height { get; set; }
    public int Width { get; set; }
    public Point BallPos { get; set; }
    public int ScoreTeamA { get; set; }
    public int ScoreTeamB { get; set; }
    private readonly string[,] _grid;
    public Court(int h, int w)
    {
        Height = h;
        Width = w;
        _grid = new string[h,w];
    }
    ...
}
```

Figure 6 ToWallGrid Method

```
public PathNode[,] ToWallGrid()
{
    var wallGrid = new PathNode[Height, Width];

    for (var i = 0; i < Height; i++)
    {
        for (var j = 0; j < Width; j++)
        {
            wallGrid[i, j] = new PathNode
            {
                IsWall = !string.IsNullOrEmpty(_grid[i,j]),
                X = i,
                Y = j,
            };
        }
    }
    return wallGrid;
}
```

order to check any graphical details. The Start button, as expected, starts the game; analogous for the Stop button.

The yellow squares indicate the baskets and the white circle represents the ball. Let's start analyzing the Court class, which corresponds to the basketball court; it contains the listed fields in **Figure 5**.

Fields Height and Width are self-explanatory. BallPos indicates the position of the ball on the court. ScoreTeamA and ScoreTeamB indicate the score for each player and \_grid is the string matrix containing the logic of the court. If player A1 is on cell (0,0) and is in possession of the ball, then value \_grid [0, 0] = A1,B. The same applies for B1, hence, the possible values of any cell on the grid are A1, B1, A1,B and B1,B. Methods, indexers implemented in this class

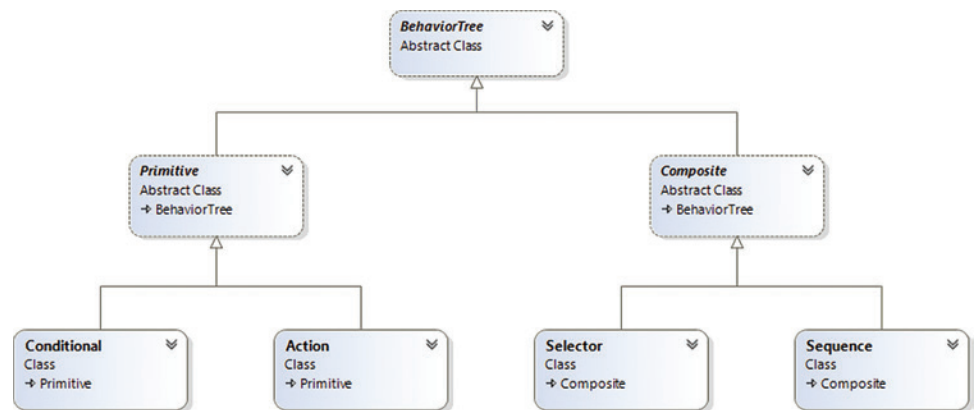


Figure 7 BehaviorTree Class Structure

are the following (Indexer provides indexing of elements on the grid; it also updates the position of the ball on the court):

```
public string this[int i, int j]
{
    get { return _grid[i, j]; }
    set
    {
        _grid[i, j] = System.String.Format("{0}", value);
        if (IsBall(value))
            BallPos = new Point(i, j);
    }
}
```

The IsBall method determines whether a given string contains the ball:

```
private bool IsBall(string s)
{
    return s.Split(',').Contains("B");
}
```

The IsEmpty method determines whether a cell on the grid is empty:

```
public bool IsEmpty(int i, int j)
{
    return string.IsNullOrEmpty(_grid[i, j]);
}
```

Finally, the ToWallGrid method returns a PathNode matrix, as shown in **Figure 6**; it will be used in the Path Finding algorithm to be explained shortly.

In this method, the wallGrid matrix will indicate whether a given cell will be considered as an obstacle or not, as mentioned, before it serves the Path Finding algorithm.

The BehaviorTree class and all of its descendants are structured according to the **Figure 7** diagram.

The code for every BehaviorTree-related class is presented in **Figure 8**.

Given the fact that every class is straightforward and the code self-explanatory, no description of them will be provided; you can easily see what the purpose of each class is and how it links to previously explained concepts.

The most significant class in the application is the Player class representing the agent itself and encapsulating its behavior and all of the AI code. This behavior has been divided into offensive and defensive; the first has been modeled through an FSM and the latter has been modeled using a simple BT like the one shown in **Figure 3**. A Player contains the listed fields shown in **Figure 9**.

Position defines the position of the player on the court. Scoring-Basket is the location where he should score on the court. Path is a list of PathNodes used to find the shortest path from one initial

**Figure 8 Code for Every BehaviorTree Related Class**

```
public abstract class BehaviorTree
{
    public List<BehaviorTree> Children { get; set; }
    public BehaviorTree Value { get; set; }
    public Court Court { get; set; }

    protected BehaviorTree(Court court)
    {
        Court = court;
    }

    public abstract bool Exec();
}

public abstract class Primitive : BehaviorTree
{
    protected Primitive(Court court) : base(court)
    {
    }
}

public class Action : Primitive
{
    public delegate bool Act();
    public Act Function { get; set; }

    public Action(Court court) : base(court)
    {
    }

    public override bool Exec()
    {
        return Function();
    }
}

public class Conditional : Primitive
{
    public delegate bool Pred();
    public Pred Predicate { get; set; }

    public Conditional(Court court)
        : base(court)
    {
    }

    public override bool Exec()
    {
        return Predicate();
    }
}

}

public abstract class Composite : BehaviorTree
{
    protected Composite(Court court) : base(court)
    {
    }

    public class Sequence : Composite
    {
        public Sequence(Court court)
            : base(court)
        {
        }

        public List<int> Order { get; set; }

        public override bool Exec()
        {
            if (Order.Count != Children.Count)
                throw new Exception("Order and children count must be the same");

            foreach (var i in Order)
            {
                if (!Children[i].Exec())
                    return false;
            }

            return true;
        }
    }

    public class Selector : Composite
    {
        public Selector(Court court)
            : base(court)
        {
        }

        public int Selection { get; set; }

        public override bool Exec()
        {
            return Children[Selection].Exec();
        }
    }
}
```

Figure 9 A Player Class Code Field

```
public class Player
{
    public Point Position { get; set; }
    public string Name { get; set; }
    public int Number { get; set; }
    public int Speed { get; set; }
    public double ShootingAccuracy { get; set; }
    public bool BallPossession { get; set; }
    public Point ScoringBasket { get; set; }
    public LinkedList<PathNode> Path;
    private readonly Court _court;
    private readonly Random _random;
    public Player(Court court, Point basket)
    {
        ScoringBasket = new Point(basket.X, basket.Y);
        _court = court;
        Path = new LinkedList<PathNode>();
        _random = new Random();
    }
}
```

Figure 10 Agent's Function

```
private IEnumerable<Percept> GetPercepts()
{
    var result = new List<Percept>();

    if (IsCloseToBasket())
        result.Add(Percept.CloseToBasket);
    if (IsBallLoose())
        result.Add(Percept.BallLoose);
    if (IsCloseToBasket())
        result.Add(Percept.CloseToBasket);
    if (BallPossession)
        result.Add(Percept.BallInPossession);
    else
        result.Add(Percept.OnDefense);

    return result;
}
```

Figure 11 The Move Method

```
private void Move(Direction direction)
{
    if (!FeasibleMoves().Contains(direction))
        return;

    _court[Position.X, Position.Y] = "";

    switch (direction)
    {
        case Direction.Left:
            Position = new Point(Position.X, Position.Y - 1);
            break;
        case Direction.Right:
            Position = new Point(Position.X, Position.Y + 1);
            break;
        case Direction.Up:
            Position = new Point(Position.X - 1, Position.Y);
            break;
        case Direction.Down:
            Position = new Point(Position.X + 1, Position.Y);
            break;
    }

    // To write his correct value on the grid
    _court[Position.X, Position.Y] =
        (_court.BallPos.X == Position.X && _court.BallPos.Y == Position.Y) ||
        BallPossession
        ? Name + ",B"
        : Name;

    if (_court[Position.X, Position.Y].Split(',').Contains("B"))
        BallPossession = true;
}
```

point to another considering obstacles on the way and `_random` is a `Random` object used to get the probability of making a shot. The remaining fields are self-explanatory.

This class uses the following enums:

```
public enum Percept
{
    Guarded,
    CloseToBasket,
    BallLoose,
    BallInPossession,
    OnDefense,
    None
}

public enum Direction
{
    Up, Down, Left, Right
}
```

The `Player` class is divided into Predicates methods and Action methods. There are three Predicates:

```
private bool IsBallLoose()
{
    return _court[_court.BallPos.X, _court.BallPos.Y] == "B";
}
```

```
private bool IsCloseToBasket()
{
    return Math.Abs(Position.X - ScoringBasket.X) <= 1 && Math.Abs(
        Position.Y - ScoringBasket.Y) <= 1;
}
```

The `IsBallLoose` method simply determines if the ball is loose on the court; the `IsCloseToBasket` method determines whether the player is close to his scoring basket:

```
private bool IsOpponentReachable()
{
    var opponents = FindOpponents();
    var factor = ScoringBasket.Y == 0 ? 1 : -1;

    foreach (var opponent in opponents)
    {
        if ((Position.Y - opponent.Y) * factor >= 0)
            return true;
    }

    return false;
}
```

`IsOpponentReachable` indicates the possibility of reaching one of the opponents on court; the factor variable aids on deciding if an opponent is in a reachable position. Reachable means that the opponent hasn't passed (while in offensive mode) the player's column when moving toward his scoring basket.

Before looking at the Actions block code, let's analyze two methods that are called immediately after an agent executes an action:

```
public void Action()
{
    var percepts = GetPercepts();

    if (percepts.Contains(Percept.CloseToBasket))
        Shoot();
    else if (percepts.Contains(Percept.BallLoose))
        MoveToBall();
    else if (percepts.Contains(Percept.BallInPossession))
        MoveToBasket();
    else if (percepts.Contains(Percept.OnDefense))
        Defend();
}
```

This method represents the agent's function; it gets a set of percepts and reacts or decides which action to take looking at the percepts, as shown in **Figure 10**.

The `GetPercepts` method, which relies on several predicates, returns the set of percepts that are eventually used to decide which action to take.



First, the FeasibleMoves method acts as a filter; once the agent has decided a move in a certain direction, it checks the FeasibleMoves list of directions and sees whether the direction the agent is about to make is actually possible; if not then no action is taken. The Move method that includes a call to the FeasibleMoves method is shown in **Figure 11**.

The MoveToBall method moves the player toward the ball in case it's loose on the court:

```
privatevoid MoveToBall()
{
    var ballPos = _court.BallPos;

    if (ballPos.X == Position.X)
        Move(ballPos.Y > Position.Y ? Direction.Right : Direction.Left);
    elseif (ballPos.Y == Position.Y)
        Move(ballPos.X > Position.X ? Direction.Up : Direction.Down);
}
```

As shown in **Figure 12**, the MoveToBasket represents a distinctive method in the application as it's the only one that includes planning and makes the agent hybrid (reactive and deliberative).

**Figure 12 The MoveToBasket Method**

```
privatevoid MoveToBasket()
{
    if (Path.Count == 0)
    {
        Path = newLinkedList<PathNode>(PathFinding(Position, ScoringBasket));
        Path.RemoveFirst();
    }

    // Already have a strategy
    if (Path.Count > 0)
    {
        var nextMove = Path.First();
        Path.RemoveFirst();

        // Check if move still available
        if (string.IsNullOrEmpty(_court[nextMove.X, nextMove.Y]))
            MoveDecision(nextMove);
        else
            Path.Clear();
    }
}
```

**Figure 13 Defensive Behavior Using a Behavior Tree**

```
privatevoid Defend()
{
    DefensiveBehavior(_court).Exec();
}

privateBehaviorTree DefensiveBehavior(Court court)
{
    var isReachableNode = newConditional(court)
    {
        Predicate = IsOpponentReachable
    };

    var blockNode = newAction(court)
    {
        Function = BlockPath
    };

    var defenseBehavior = newSequence(court)
    {
        Order = newList<int> {0,1},
        Children = newList<BehaviorTree>
        {
            isReachableNode,
            blockNode
        }
    };

    return defenseBehavior;
}
```

**Figure 14 The BlockPath Method**

```
privatebool BlockPath()
{
    var closestOppPos = Closest(FindOpponents());

    // Move to same row
    if (closestOppPos.X > Position.X)
        Move(Direction.Down);
    elseif (closestOppPos.X < Position.X)
        Move(Direction.Up);
    // Move to same column
    elseif (closestOppPos.Y > Position.Y)
        Move(Direction.Right);
    elseif (closestOppPos.Y < Position.Y)
        Move(Direction.Left);

    returntrue;
}
```

MoveToBasket builds a path from the player's position up to the basket; if the plan fails or becomes unfeasible, then that path is erased and recalculated again. The PathFinding algorithm is a search algorithm; in this case, an A\* algorithm. Path finding algorithms are frequently implemented in AI and are extremely common in games.

The most significant class in the application is the Player class representing the agent itself and encapsulating its behavior and all of the AI code.

As previously mentioned, the defensive behavior is developed using a BT, as shown in **Figure 13**.

The BlockPath method represents the strategy by which a player tries to block his closest opponent path to the basket. It relies on the Closest method, which uses the Manhattan Distance to find his closest opponent, as shown in **Figure 14**.

## Wrapping Up

In this article, I explained how to develop a hybrid agent for a multi-agent basketball game. It's now up to you to incorporate new functionalities and strengthen the AI proposed. In a future article I'll address the issue of creating a learning agent for a basketball game; a learning agent will evolve over time and improve its strategies with every new play. ■

**ARNALDO PÉREZ CASTAÑO** is a computer scientist based in Cuba. He's the author of a series of programming books—"JavaScript Fácil," "HTML y CSS Fácil," and "Python Fácil" (Marco Polo S.A.)—and writes for VisualStudioMagazine.com and Smashing Magazine. His expertise includes Visual Basic, C#, .NET Framework and artificial intelligence, and he offers his services as a freelancer throughnubelo.com. Cinema and music are some of his passions. Contact him at [arnaldo.skywalker@gmail.com](mailto:arnaldo.skywalker@gmail.com).

**THANKS** to the following Microsoft technical expert for reviewing this article: James McCaffrey

Orlando  
2016

ROYAL PACIFIC RESORT AT  
UNIVERSAL ORLANDO

DEC  
5-9



# Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

## *Journey into Code*

Join us as we journey into real-world, practical education and training on the Microsoft Platform. Visual Studio Live! (VSLive!™) returns to warm, sunny Orlando for the conference more developers rely on to expand their .NET skills and the ability to build better applications.



[twitter.com/live360](https://twitter.com/live360)  
[@live360](https://twitter.com/live360)



[facebook.com](https://facebook.com)  
Search "Live 360"



[linkedin.com](https://linkedin.com)  
Join the "Live! 360" group!



EVENT PARTNERS



Microsoft

**Magenic**

PLATINUM SPONSOR



GOLD SPONSOR



SUPPORTED BY







TECH EVENTS WITH PERSPECTIVE

## 6 Great Conferences 1 Great Price

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to five (5) other co-located events at no additional cost:

SQL Server **LIVE!**  
TRAINING FOR DBAs AND IT PROS

TECHMENTOR  
IN-DEPTH TRAINING FOR IT PROS

Office & SharePoint **LIVE!**  
ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

ModernApps **LIVE!**  
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

**NEW!** APPDEV TRENDS  
ENTERPRISE FOCUSED. CODE DRIVEN.

Six (6) events and hundreds of sessions to choose from - mix and match sessions to create your own, custom event line-up - it's like no other conference available today!

### Whether you are an

- Engineer
- Developer
- Programmer
- Software Architect
- Software Designer

You will walk away from this event having expanded your .NET skills and the ability to build better applications.

### REGISTER TODAY AND SAVE \$500!



Use promo code  
ORLJUL2 by August 31

Scan the QR code to  
register or for more  
event details.

PRODUCED BY

Redmond  
MAGAZINE

VIRTUALIZATION  
MAGAZINE

Visual Studio  
MAGAZINE

IIOS MEDIA

[VSLIVE.COM/ORLANDO](http://VSLIVE.COM/ORLANDO)



## Matrix Inversion Using C#

One of the most fundamental techniques in machine learning (ML) software systems is matrix inversion. For reasons that aren't clear to me, the Microsoft .NET Framework doesn't seem to have a matrix inversion method (or if there is such a method, it's very well hidden). In this article I present and explain the code for a matrix inversion method that uses an algorithm called Crout's LU decomposition.

Let me be the first to admit that matrix inversion isn't a very flashy topic. But if you want to create ML systems without relying on external libraries, having a matrix inversion method is essential because matrix inversion is used by dozens of important ML algorithms.

A good way to see where this article is headed is to take a look at the demo program in **Figure 1**.

The demo begins by setting up and displaying a 4x4 (4 rows, 4 columns) matrix m:

```
3.0 7.0 2.0 5.0
1.0 8.0 4.0 2.0
2.0 1.0 9.0 3.0
5.0 4.0 7.0 1.0
```

It then calculates the inverse of the matrix using a program-defined method and displays the result:

```
0.097 -0.183 -0.115 0.224
-0.019 0.146 -0.068 0.010
-0.087 0.064 0.103 -0.002
0.204 -0.120 0.123 -0.147
```

That's pretty much it. But as you'll see shortly, matrix inversion is surprisingly tricky. Next, the demo verifies that the calculated inverse is correct by multiplying the original matrix times the inverse:

```
1.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0
0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0
```

The result of the multiplication is the identity matrix (1.0 values on the diagonal, 0.0 values elsewhere) indicating the inverse result is correct.

Behind the scenes, the matrix inversion method uses a technique called matrix decomposition. Decomposition factors a matrix into two matrices, called L (lower) and U (upper), that when multiplied together give the original matrix, but with some of the rows rearranged. The demo displays the decomposition:

```
5.000 4.000 7.000 1.000
0.200 7.200 2.600 1.800
0.400 -0.083 6.417 2.750
0.600 0.639 -0.602 4.905
```

The decomposition actually contains both the L and U matrices. The L matrix consists of the values in the lower-left part of the combined LU matrix, with dummy 1.0 values on the diagonal and 0.0 values in the upper part:

```
1.000 0.000 0.000 0.000
0.200 1.000 0.000 0.000
0.400 -0.083 1.000 0.000
0.600 0.639 -0.602 1.000
```

The U matrix consists of the values in the upper-right part and the diagonal of the combined LU matrix:

```
5.000 4.000 7.000 1.000
0.000 7.200 2.600 1.800
0.000 0.000 6.417 2.750
0.000 0.000 0.000 4.905
```

```
file:///C:/MatrixInverse/bin/Debug/MatrixInverse.EXE
Begin matrix inverse using Crout LU decomp demo
Original matrix m is
3.000 7.000 2.000 5.000
1.000 8.000 4.000 2.000
2.000 1.000 9.000 3.000
5.000 4.000 7.000 1.000
Inverse matrix inv is
0.097 -0.183 -0.115 0.224
-0.019 0.146 -0.068 0.010
-0.087 0.064 0.103 -0.002
0.204 -0.120 0.123 -0.147
The product of m * inv is
1.000 0.000 0.000 0.000
0.000 1.000 0.000 0.000
0.000 0.000 1.000 0.000
0.000 0.000 0.000 1.000
=====
The combined lower-upper decomposition of m is
5.000 4.000 7.000 1.000
0.200 7.200 2.600 1.800
0.400 -0.083 6.417 2.750
0.600 0.639 -0.602 4.905
The lower part of LUM is
1.000 0.000 0.000 0.000
0.200 1.000 0.000 0.000
0.400 -0.083 1.000 0.000
0.600 0.639 -0.602 1.000
The upper part of LUM is
5.000 4.000 7.000 1.000
0.000 7.200 2.600 1.800
0.000 0.000 6.417 2.750
0.000 0.000 0.000 4.905
The perm[] array is
3 1 2 0
The product of lower * upper is
5.000 4.000 7.000 1.000
1.000 8.000 4.000 2.000
2.000 1.000 9.000 3.000
3.000 7.000 2.000 5.000
End matrix inverse demo
```

Figure 1 Matrix Inversion Demo

Code download available at [msdn.com/magazine/0716magcode](http://msdn.com/magazine/0716magcode).



The demo verifies that the LU decomposition is correct by multiplying the L and U matrices and displaying the result:

```
5.0 4.0 7.0 1.0
1.0 8.0 4.0 2.0
2.0 1.0 9.0 3.0
3.0 7.0 2.0 5.0
```

If you compare  $L \cdot U$  with the original matrix  $m$ , you'll see that  $L \cdot U$  is almost the same as  $m$ , but the rows of  $L \cdot U$  have been permuted (rearranged). The row permutation information is:

```
3 1 2 0
```

This means row[0] of  $m$  is at row[3] of  $L \cdot U$ ; row[1] of  $m$  is at row[1] of  $L \cdot U$ ; row[2] of  $m$  is at row[2] of  $L \cdot U$ ; and row[3] of  $m$  is at row[0] of  $L \cdot U$ .

## Understanding Matrix Inversion

In normal arithmetic, the inverse of a number  $z$  is a number that when multiplied by  $z$  gives 1. For example, if  $z = 3$ , the inverse of  $z$  is  $1/3 = 0.33$  because  $3 * (1/3) = 1$ .

As it turns out, there is a scalar value called the determinant of a matrix. If the determinant of a matrix is zero, then the matrix doesn't have an inverse.

Matrix inversion extends this idea. The inverse of an  $n \times n$  (called a "square matrix" because the number of rows equals the number of columns) matrix  $m$  is a matrix  $mi$  such that  $m * mi = I$  where  $I$  is the identity matrix (1.0s on the diagonal, 0.0s elsewhere).

Not all matrices have an inverse. As it turns out, there is a scalar value called the determinant of a matrix. If the determinant of a matrix is zero, then the matrix doesn't have an inverse.

Note that to fully understand matrix inversion, you must understand matrix multiplication. Matrix multiplication is best explained by example. Take a look at the example in **Figure 2**. The value at cell  $[r][c]$  of the result matrix is the product of the values in row  $r$  of the first matrix and the values in column  $c$  of the second matrix.

When finding the inverse of a matrix, you work only with square matrices, but matrix multiplication can be applied to matrices with different shapes. In these situations the matrices must be what's called conformable. If matrix  $A$  has shape  $axn$  and matrix  $B$  has shape  $nxb$ , the result of multiplication has shape  $axb$ . The number

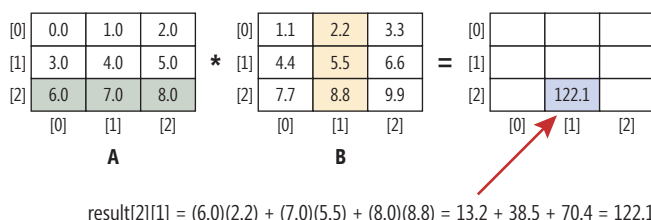


Figure 2 Matrix Multiplication

of columns in the first matrix must equal the number of rows in the second matrix.

The demo program implements matrix multiplication with method `MatrixProduct` and helper method `MatrixCreate`, as shown in **Figure 3**. The demo uses a brute force approach, but because the calculation of each cell in the result matrix is independent, matrix multiplication could be performed in parallel using the `Parallel.For` method from the .NET Task Parallel Library.

## The Demo Program

I coded the demo program using C#, but you should have no difficulty porting the code to another language, such as Visual Basic or Python, if you wish. The demo code is too long to present in its entirety, but the complete code is available in the download that accompanies this article. The code is also available at [quaetrix.com/Matrix/code.html](http://quaetrix.com/Matrix/code.html) (the URL is case-sensitive).

To create the demo program, I launched Visual Studio and created a new C# console application named `MatrixInverse`. The demo program has no significant .NET Framework dependencies so any version of Visual Studio will work. After the template code loaded, in the Solution Explorer window I right-clicked on file `Program.cs` and renamed it to the more descriptive `MatrixInverseProgram.cs` and Visual Studio then automatically renamed class `Program` for me.

At the top of the editor window I deleted all using statements that referenced unnecessary namespaces, leaving just the one reference to the top-level `System` namespace.

The `Main` method begins by setting up a matrix to invert:

```
Console.WriteLine("Begin matrix inverse demo");
double[][] m = MatrixCreate(4, 4);
m[0][0] = 3; m[0][1] = 7; m[0][2] = 2; m[0][3] = 5;
m[1][0] = 1; m[1][1] = 8; m[1][2] = 4; m[1][3] = 2;
m[2][0] = 2; m[2][1] = 1; m[2][2] = 9; m[2][3] = 3;
m[3][0] = 5; m[3][1] = 4; m[3][2] = 7; m[3][3] = 1;
```

In many cases, your source data will be stored in a text file so you'll have to write a helper method to load a matrix from the file. The demo uses an array-of-arrays-style matrix. Unlike most programming

Figure 3 Matrix Multiplication

```
static double[][] MatrixCreate(int rows, int cols)
{
    double[][] result = new double[rows][];
    for (int i = 0; i < rows; ++i)
        result[i] = new double[cols];
    return result;
}

static double[][] MatrixProduct(double[][] matrixA,
    double[][] matrixB)
{
    int aRows = matrixA.Length;
    int aCols = matrixA[0].Length;
    int bRows = matrixB.Length;
    int bCols = matrixB[0].Length;
    if (aCols != bRows)
        throw new Exception("Non-conformable matrices");

    double[][] result = MatrixCreate(aRows, bCols);

    for (int i = 0; i < aRows; ++i)
        for (int j = 0; j < bCols; ++j)
            for (int k = 0; k < aCols; ++k)
                result[i][j] += matrixA[i][k] *
                    matrixB[k][j];

    return result;
}
```

languages, C# supports a true n-dimensional matrix type, but I prefer using the standard array-of-arrays approach.

Next, Main displays the matrix m, and then computes and displays the inverse:

```
Console.WriteLine("Original matrix m is ");
Console.WriteLine(MatrixAsString(m));
double[][] inv = MatrixInverse(m);
Console.WriteLine("Inverse matrix inv is ");
Console.WriteLine(MatrixAsString(inv));
```

All the work is performed by method `MatrixInverse`. Helper method `MatrixAsString` returns a string representation of a matrix:

```
static string MatrixAsString(double[][] matrix)
{
    string s = "";
    for (int i = 0; i < matrix.Length; ++i) {
        for (int j = 0; j < matrix[i].Length; ++j)
            s += matrix[i][j].ToString("F3").PadLeft(8) + " ";
        s += Environment.NewLine;
    }
    return s;
}
```

Here the number of decimals (3) and value width (8) are hard-coded for simplicity. A more general approach would pass those values as input parameters. Next, the demo multiplies the original matrix and the inverse matrix in order to verify that the result is the identity matrix:

```
double[][] prod = MatrixProduct(m, inv);
Console.WriteLine("The product of m * inv is ");
Console.WriteLine(MatrixAsString(prod));
```

It should be clear that method `MatrixInverse` is essentially a wrapper around methods `MatrixDecompose` and `Helper`, which do most of the work.

In this version, you have to visually verify that the result is the identity matrix. A more sophisticated approach would be to write a method that accepts a matrix and returns true if the matrix is an identity matrix, subject to some small difference ( $1.0e-5$  is typical) in cell values.

Next, the demo illustrates some of the behind-the-scenes work by decomposing the original matrix:

```
double[][] lum;
int[] perm;
int toggle = MatrixDecompose(m, out lum, out perm);
Console.WriteLine("The decomposition is");
Console.WriteLine(MatrixAsString(lum));
```

The calling signature of method `MatrixDecompose` might appear a bit unusual to you. The explicit return value is either +1 or -1 depending on the number of row permutations there were (even or odd, respectively). The toggle return value isn't used by the demo, but is needed if you want to compute the determinant of the matrix, which tells you if the inverse of a matrix exists, as I'll explain shortly.

The lum out parameter is the combined LU (lower-upper) decomposition. The perm out parameter is an array of integer values that encode how the rows have been permuted.

Next, the demo extracts the lower and upper matrices from the combined LU matrix and displays them:

```
double[][] lower = ExtractLower(lum);
double[][] upper = ExtractUpper(lum);
Console.WriteLine("The lower part of LUM is");
Console.WriteLine(MatrixAsString(lower));
Console.WriteLine("The upper part of LUM is");
Console.WriteLine(MatrixAsString(upper));
```

Helper methods `ExtractLower` and `ExtractUpper` really aren't needed to perform matrix inversion, but are used to illustrate how matrix decomposition works.

The demo program concludes by displaying the row permutation information, multiplying the lower and upper decomposition matrices, and displaying the result:

```
Console.WriteLine("The perm[] array is");
ShowVector(perm);
double[][] lowTimesUp = MatrixProduct(lower, upper);
Console.WriteLine("The product of lower * upper is ");
Console.WriteLine(MatrixAsString(lowTimesUp));
Console.WriteLine("End matrix inverse demo");
```

Program-defined helper method `ShowVector` is just a convenience to keep the Main method clean. As noted, the result of lower \* upper is the original matrix, except that the rows of the result are permuted according to the information in the perm array.

## Method `MatrixInverse`

The definition of method `MatrixInverse` begins with:

```
static double[][] MatrixInverse(double[][] matrix)
{
    int n = matrix.Length;
    double[][] result = MatrixCreate(n, n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = matrix[i][j];
    ...
}
```

The method assumes that its input parameter does, in fact, have a matrix. This means you should check before calling, along the lines of:

```
int d = MatrixDeterminant(m);
if (d == 0.0)
    Console.WriteLine("No inverse");
else
    double[][] mi = MatrixInverse(m);
```

An alternative is to place this error-checking code inside method `MatrixInverse`, which creates a copy of the input matrix. You could also perform matrix inversion in place, which saves memory but destroys the original matrix.

Next, `MatrixInverse` decomposes the copy of the input matrix:

```
double[][] lum; // Combined lower & upper
int[] perm;
int toggle;
toggle = MatrixDecompose(matrix, out lum, out perm);
```

It may seem strange to go to all the trouble of decomposing a matrix in order to compute its inverse, but trust me, this approach is much easier than inverting a matrix directly.

Next, the demo computes the inverse using yet another helper method named `Helper`:

```
double[] b = new double[n];
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
        if (i == perm[j]) b[j] = 1.0;
        else b[j] = 0.0;

    double[] x = Helper(lum, b); //
    for (int j = 0; j < n; ++j)
        result[j][i] = x[j];
}
```

This code is very subtle. Fortunately, you won't ever have to modify this part of the code.

Method `MatrixInverse` concludes by returning the inverse:

```
...
return result;
}
```

It should be clear that method `MatrixInverse` is essentially a wrapper around methods `MatrixDecompose` and `Helper`, which do most of the work. The code for those two methods are in the download that accompanies this article.

## The Determinant of a Matrix

If the determinant of a matrix is zero, then the matrix doesn't have an inverse. Suppose a 3x3 matrix is:

```
1.0  4.0  0.0
3.0  2.0  5.0
7.0  8.0  6.0
```

The determinant of the matrix is:

```
+1.0 * [(2.0)(6.0) - (5.0)(8.0)]
-4.0 * [(3.0)(6.0) - (5.0)(7.0)]
+0.0 * [(3.0)(8.0) - (2.0)(7.0)]
```

```
= +1.0 * (-28.0) -4.0 * (-17.0) = -28.0 + 68.0 = 40.0
```

Every square matrix has a determinant. For matrices with shapes larger than 3x3, calculating the determinant is surprisingly difficult. However, as it turns out, if you decompose a matrix you can use the combined lower-upper result to calculate the determinant rather easily by multiplying the diagonal elements of the result. The demo program defines a method `MatrixDeterminant` as:

```
static double MatrixDeterminant(double[][] matrix)
{
    double[][] lum;
    int[] perm;
    int toggle = MatrixDecompose(matrix, out lum,
    out perm);
    double result = toggle;
    for (int i = 0; i < lum.Length; ++i)
        result *= lum[i][i];
    return result;
}
```

## Wrapping Up

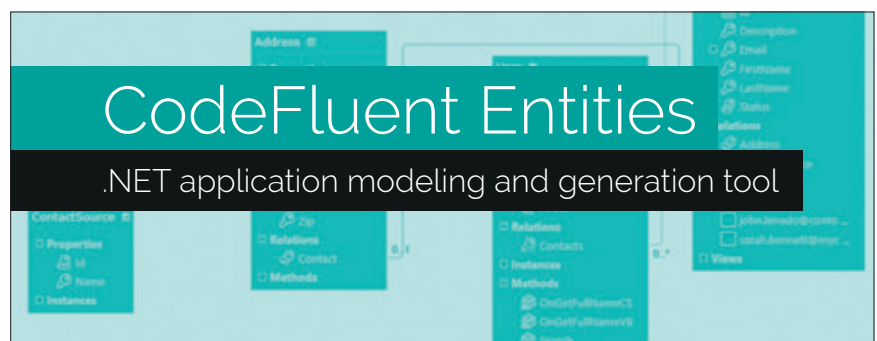
The key to efficient matrix inversion is matrix decomposition. There are several algorithms that decompose a matrix. The demo code uses a technique called Crout's algorithm. A common alternative is Doolittle's algorithm. I used to prefer Doolittle's algorithm because it's a bit simpler than Crout's, but I now favor Crout's algorithm because it has fewer places to fail.

I've always wondered why the .NET Framework doesn't have a method that calculates the inverse of a matrix. To be sure, matrix inversion is very, very tricky. I tested the code presented in this article by randomly generating 100 million square matrices with shapes between 2x2 and 50x50, calculating

the inverses, and programmatically verifying that the product of the inverse and the original matrix was the identity matrix of the appropriate size. ■

**DR. JAMES MCCAFFREY** works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. Dr. McCaffrey can be reached at [jammc@microsoft.com](mailto:jammc@microsoft.com).

**THANKS** to the following Microsoft technical experts who reviewed this article: Chris Lee and Kirk Olynky



Create **high-quality software**  
always aligned with your business rules!

Ultimate Edition now only USD699 / year  
Free edition still available

If you like to start from a visual model rather than dive in code first, if you like to focus on business code rather than continuously write plumbing code, give it a try.

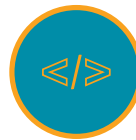
Code and model are kept in sync and leverage best practices to ensure application maintainability and performance as well as developer productivity.

CodeFluent Entities is fully integrated in Visual Studio 2008 to 2015.



### MODEL YOUR BUSINESS

Leverage the graphical modeler to define your business model in a centralized place



### DEFINE YOUR FOUNDATIONS

Generate a direct translation of your model into ready-to-use data and business layers



### DEVELOP YOUR APPLICATIONS

Build modern applications by focusing on your high-value custom code

soft<luent

Innovative software company founded in 2005 by Microsoft veterans

SoftFluent - 2018 156th Avenue NE Bellevue, WA 98007 USA - [info@softfluent.com](mailto:info@softfluent.com) - [www.softfluent.com](http://www.softfluent.com)

Copyright © 2005 - 2016, SoftFluent SAS. All rights reserved.

Microsoft, Visual Studio and Excel® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.





## How To Be MEAN: Let's Be DEAN

Welcome back again, MEANers. Or, rather, for this month, “DEAN”ers.

One of the things that makes conversations and architecture around the MEAN stack so compelling is that the MEAN stack is intrinsically flexible—you can replace components of the stack with other, more-or-less-equivalent parts and create a new stack that can address corporate/business needs without surrendering the essence of the architecture. As a demonstration of that concept, in this column I’m going to experiment with replacing MongoDB with Microsoft Azure DocumentDB (hence, the “D” in place of the “M”).

### MongoDB vs. DocumentDB

For those who haven’t spent much time getting familiar with DocumentDB, I highly recommend looking at some of the existing resources available, including Julie Lerman’s June 2015 Data Points column ([msdn.com/magazine/mt147238](http://msdn.com/magazine/mt147238)). Lerman gives a great overview of DocumentDB, including one of its most interesting features, server-side code execution (yes, as in stored procedures, written in JavaScript). If you’ve never glanced at DocumentDB, or need a quick intro, do a quick read before continuing.

On the surface, MongoDB and DocumentDB are similar creatures. They’re both document-oriented databases, using JSON as the principal document representation format. They each extend JSON to incorporate some additional data types. They each think about data in terms of collections of documents, and use that as the principal aggregation scheme (instead of tables). They’re intrinsically “non-relational” in that the database doesn’t do any verification of document identifiers used to link a document to another document. For that matter, they’re also both “schema-free,” in that a document’s contents are entirely in the hands of the developer, and the same document structure isn’t enforced across all the elements of a collection.

Figure 1 Config.js in the Existing Code Base

```
module.exports = {
  mongoServer : "localhost",
  mongoPort : "27017",
  docdbServer : "https://dean-db.documents.azure.com:443/",
  docdbKey :
    "gzk030R7xC9629Cm10AUirYg8n2sqLF300xtrV18JT
    ANNM1zV1KL14VEShJyB70jEtdmwnUuc4nRYyHhxsjQjQ=="
};

if (process.env["ENV"] === "prod") {
  module.exports.mongoServer = "ds054308.mongolab.com";
  module.exports.mongoPort = "54308";
  module.exports.docdbServer = process.env["DOCDB_HOST"];
  module.exports.docdbKey = process.env["DOCDB_KEY"];
}
```

Therefore, this should be a simple swap.

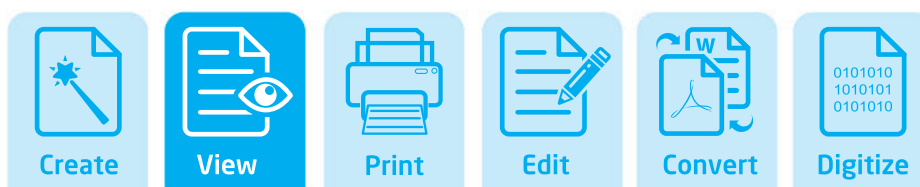
However, aside from some deeper differences you’ll discover in a bit, right away there’s one principal difference: While MongoDB can be downloaded and run locally off a laptop, DocumentDB is only available through an Azure subscription, so the first step toward DEAN is the simple task of creating a DocumentDB instance on an Azure account. This is described in several other places in detail (including in the DocumentDB documentation online), so I won’t repeat the process here except to summarize. Go to the Azure Management Portal, create a new Azure DocumentDb resource, give it a unique name (I called mine dean-db) and punch the big blue Create button. The portal will churn for a few minutes and Azure will create your DocumentDB instance in the cloud.

As a demonstration of that concept, in this column I’m going to experiment with replacing MongoDB with Microsoft Azure DocumentDB (hence, the “D” in place of the “M”).

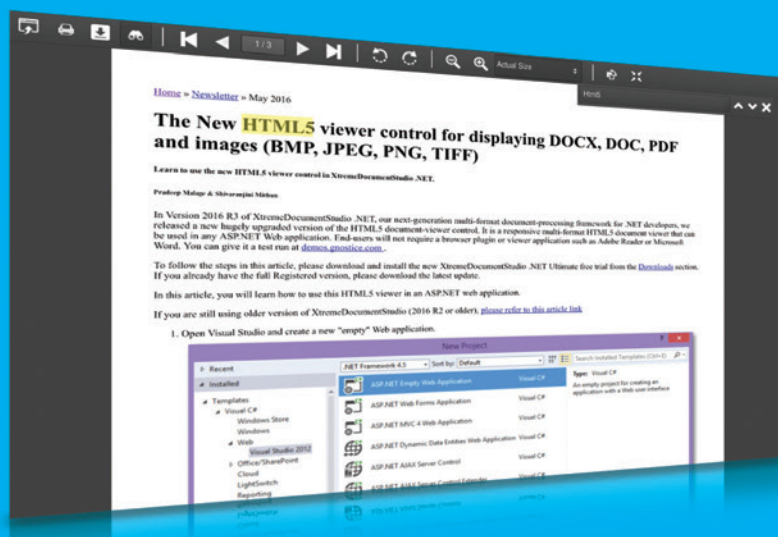
Before you’re done with the portal, there are a couple pieces of information you’re going to need later in the code. So let’s take a second and jot them down now. In particular, you’ll need the URL to connect to, and the authorization key to ensure it’s you. In the Azure Management Portal, this is called a PRIMARY KEY (there’s a second one called SECONDARY KEY), and as of this writing, it’s displayed in the All Settings tab, under Keys. These keys are shared secrets, so make sure not to give it out to anybody—in particular, if the code is going into a public source code repository (a la GitHub), make sure the key isn’t part of the checked-in source code. (Conventionally, Node.js applications configure this as an environment variable and use the application entry point code to pick this up as part of its startup; you’ve done similar to this in previous iterations of this code base, so this shouldn’t be a surprise.) If a key is compromised in any way (as in, you discover that the key somehow made it out into the public), make sure to regenerate the keys



# Document Technology for Everybody



## Introducing The New Interactive Multi-Format HTML5 Document Viewer



- Responsive HTML5 control, automatically adjusts for Mobile, Desktop and Pad/Tablets.
- 100% Independent. No browser plug-ins required. No ActiveX. No Office Automation.
- Single viewer for PDF, Office documents, text files and Images.
- Text Search, On-the-fly OCR on images, PDF form-filling, and more...
- Full-fledged client-side JavaScript to configure and perform all operations.
- TypeScript Support.
- Simple, straightforward licensing.



XtremeDocumentStudio  
.NET



StarDocs API  
Server



XtremeDocumentStudio  
Java



XtremeDocumentStudio  
Delphi

Figure 2 Querying DocumentDB for a List of Databases

```
docClient.queryDatabases({
  query: 'SELECT * FROM root r WHERE r.id = @id',
  parameters: [
    {
      name: '@id',
      value: 'conferencedb'
    }
  ]
}).toArray(function (err, results) {
  if (err) {
    handleError(err);
  }

  if (results.length === 0) {
    // No error occurred, but there were no results returned
    // indicating no database exists matching the query
    // so, explicitly return null
    debug("No results found");
  } else {
    // Found a database, so return it
    debug('Found a database:', results[0]);
    var docDB = results[0];
  }
});
```

and replace them. In fact, it's probably a good idea to cycle them every 30 or 60 days just on principle.

This means that the `config.js` in the existing code base now looks like **Figure 1**.

(By the way, I've already cycled the keys on my repository, so displaying the key here is just for show, to give you an idea of what it'll look like.)

## Hello, from Node.js

The next step is predictable—you need the Node.js module for accessing the DocumentDB instance, and you do so using “npm install --save documentdb.” Naturally, you'll need to “require” it in the `app.js` code, like so:

```
var express = require('express'),
    bodyParser = require('body-parser'),
    debug = require('debug')('app'),
    edge = require('edge'),
    documentdb = require('documentdb'),
    ...;

// Go get our configuration settings
var config = require('./config.js');
debug("Mongo is available at ", config.mongoServer, ":", config.mongoPort);
debug("DocDB is available at ", config.docdbServer);
```

From here, opening a connection is simply a matter of constructing a `DocumentClient` object using the server and authorization key, like so:

```
// Connect to DocumentDB
var docDB = documentdb.DocumentClient(config.docdbServer, {
  masterKey: config.docdbKey
});
```

Now, of course, the bigger question is: What do you do with it, once you have it open?

## Using DocumentDB

Like MongoDB, DocumentDB uses a JSON-based format for a schema, and collections of documents, so storing data there (such as “presentations”) involves the same kinds of operations as you've seen with MongoDB in the past. However, the API is a little different, probably owing to some cultural differences between how Microsoft likes to design things and how things sort of evolve out of the open source world.

(By the way, the best DocumentDB/Node.js reference I've found thus far is the set of samples on the Azure Documentation page [[bit.ly/1TkqXaP](http://bit.ly/1TkqXaP)]. It's a collection of direct links to Microsoft-authored sample projects stored on GitHub, as well as links to the documentation generated out of the DocumentDB Node.js API source code, and in lieu of anything more official, I've been using it as my go-to reference for DocumentDB APIs.)

Microsoft is clearly marking DocumentDB as the compromise choice between SQL Server and MongoDB (or other document databases), and that's not a bad place to be.

For starters, once the database client object is constructed, you need to connect to the database in question using a `databaseId`; this is the actual database, in much the same way that a MongoDB server can host multiple databases. However, unlike MongoDB, with DocumentDB this database must be created ahead of time. This can be done either programmatically (which is great for demos and DevOps scenarios) via the `createDatabase` API call, or via the Azure Management Portal (which is probably going to be the more common approach, given that this is usually a one-off operation) using the Add Database tab under the DocumentDB resource page. For simplicity's sake, the code here assumes a database of `conferencedb` already exists, presumably created in the portal earlier. Assuming the database exists, Node.js can connect to it by calling `queryDatabases`, which brings us to the next big difference in approach between MongoDB and DocumentDB, as shown in **Figure 2**.

Figure 3 Found a Database? Now Find the Collections

```
// We found a database
debug('Found a database:', results[0]);
docDB = results[0];

debug('Looking for collections:');
docClient.readCollections('dbs/conferencedb').
  toArray(function(err, colls) {
    if (err) {
      debug(err);
    }
    else {
      if (colls.length === 0) {
        debug("No collections found");
      }
      else {
        for (var c in colls) {
          debug("Found collection", colls[c]);

          if (colls[c].id === 'presentations')
            presentationColl = colls[c];
        }
      }
    }
  });
```

Figure 4 List All the Documents in a Collection

```
var getAllPresentations = function(req, res) {
  debug("Getting all presentations from DocumentDB:");

  docClient.queryDocuments("dbs/conferencedb/colls/presentations",
  {
    query: "SELECT * FROM presentations p"
  }).toArray(function (err, results) {
    if (err) res.status(500).jsonp(err);
    else res.status(200).jsonp(results);
  });
};

// ...

app.get('/presentations', getAllPresentations);
```

First, notice how the DocumentDB API uses an explicit “query specification,” even for fetching the list of databases, complete with parameters. This is a marked difference from Mongo’s “query by example” approach. Granted, you clearly don’t have to use this—you could just do inline population of the arguments via string concatenation—but as any developer who’s ever been the victim of a SQL injection attack will testify, doing parameterized queries like this is much, much safer.

The other major difference is, of course, the query language—hello, SQL, my old friend. To be fair, it’s not exactly SQL, because you don’t have tables, columns and such, but Microsoft has gone to great lengths to adapt the familiar query language over to the document-oriented world. Whether this is a “bug” or “feature” is likely to depend on the developer’s love-or-hate relationship with the relational database world, but by these (and other) decisions, Microsoft is clearly marking DocumentDB as the compromise choice between SQL Server and MongoDB (or other document databases) and that’s not a bad place to be.

Once a database is found, you need to obtain the collection, again by identifier. Like databases, collections are “formal” things, meaning they need to be either explicitly created using the `createCollection` API call or via the Azure Management Portal. Given how DocumentDB thinks of collections as tables, this is not surprising. Again, the simplest thing to do is create one via the portal, by (as of this writing) clicking on the database name in the databases tile in the DocumentDB resource tile. This will bring up a new tab, in which you can add a new collection, presentations, and then find it via that identifier, like in **Figure 3**.

Take note of the first parameter to the `readCollections` API call; if it looks like it’s specifying some kind of URI/URL-like path to the database in question, it should. Each call to the Node.js DocumentDB API uses these kinds of REST-ish identifiers to make it easy to drill directly to the collection (and, ultimately, document) desired without having to navigate a complex hierarchy.

(One important note that’s very different from MongoDB: Azure charges for its services on a per-collection basis, so where a MongoDB user will think about collections from a pure modeling standpoint, with DocumentDB, the decision to create a new collection will have a direct impact on the monthly fees you end up paying.)

Finally, to find any particular documents in that collection, you use the `queryDocuments` API. Again, you pass in the collection identifier (which, because you know the collection you’re going after, you can just embed directly in the code), and take the results

to hand it directly back as the JSON body, just like I did a few columns back for MongoDB, as shown in **Figure 4**.

Naturally, if you want to restrict the presentations to a speaker, that would be a query parameter as part of the route configuration and would become a parameter in the query specification, and so on.

## Wrapping Up

When I began this article, I cited all the ways that MongoDB and DocumentDB are similar; however, by now you can start to sense that there are actually a number of differences between the two, by way of philosophy behind their construction, if nothing else. Microsoft is clearly looking to put some “enterprise” into the document-oriented database world with DocumentDB, and given its customer base, that’s a solid move. However, one thing that’s clearly lacking (for now) is any kind of “object-ish” wrapper around the DocumentDB API, akin to what Mongoose provides to the Node.js MongoDB API. This isn’t a complicated thing for developers to build on their own, but it does represent more code that said developers would need to write and maintain over time. That is at least until the open source community either ports Mongoose, adapts it to use either MongoDB or DocumentDB (not likely), or until someone builds something similar to it but in a more specialized way than which DocumentDB approaches life.

Naturally, if you want to restrict the presentations to a speaker, that would be a query parameter as part of the route configuration and would become a parameter in the query specification, and so on.

More important, however, is to realize that the acronym “MEAN” is just that—an acronym. DocumentDB represents only one of a dozen or so kinds of persistence tools that could be the back end of an “\*EAN”-ish stack, including good ol’ SQL Server. Where “MEAN” doesn’t fit with your corporate standards, operations staff or business goals, feel free to replace the offending piece with something that’s friendlier. At the end of the day, following an architecture to the letter (literally) that creates problems for you as a developer and/or to the organization as a whole is just silly. I’m out of space for now ... happy coding! ■

---

**TED NEWARD** is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F# MVP and has authored and coauthored a dozen books. Reach him at [ted@tedneward.com](mailto:ted@tedneward.com) if you’re interested in having him come work with your team, or read his blog at [blogs.tedneward.com](http://blogs.tedneward.com).

---

**THANKS** to the following technical expert for reviewing this article:  
Shawn Wildermuth



# GIVE YOUR

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

# Microsoft HQ

# CODE A VOICE



## Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS



**VISUAL STUDIO LIVE!** is on a Campaign for Code in 2016, and we're taking a swing through our home state, rallying @ **Microsoft Headquarters** in beautiful Redmond, WA. From August 8 – 12, developers, software architects, engineers, designers and more will convene for five days of unbiased and cutting-edge education on the Microsoft Platform. Plus, you'll be at the heart of it all: eating lunch with the Blue Badges, rubbing elbows with Microsoft insiders, exploring the campus, all while expanding your development skills and the ability to create better apps!

EVENT PARTNER



PLATINUM SPONSOR



SUPPORTED BY



PRODUCED BY





# MICROSOFT HQ • AUGUST 8-12, 2016

MICROSOFT HEADQUARTERS • REDMOND, WA



TURN THE  
PAGE FOR  
MORE EVENT  
DETAILS.

CONNECT WITH VISUAL STUDIO LIVE!



[twitter.com/vslive](https://twitter.com/vslive) – @VSLive



[facebook.com](https://facebook.com) – Search "VSLive"



[linkedin.com](https://linkedin.com) – Join the  
"Visual Studio Live" group!

## TOPICS INCLUDE:

- ALM / DevOps
- Cloud Computing
- Database & Analytics
- Mobile Client
- Software Practices
- Visual Studio / .NET Framework
- Web Client
- Web Server
- Windows Client
- Microsoft-led Sessions

Have your code heard:  
register to join us today.

**SPECIAL OFFER!**  
REGISTER WITH CODE: RDMSDN1  
**AND SAVE \$400**



Scan the QR code to  
register or for more  
event details.

**MUST USE DISCOUNT  
CODE RDMSDN1**

**VSLIVE.COM/MSDN**



**Visual Studio Live!** has partnered with the Hyatt Regency Bellevue for conference attendees at a special reduced rate.



Explore the Microsoft Headquarters during **Visual Studio Live!** Redmond 2016!



## SPECIAL OFFER!

REGISTER WITH CODE: **RDMSDN1**  
AND SAVE \$400



Scan the QR code to register or for more event details.

**MUST USE DISCOUNT CODE RDMSDN1**

ALM / DevOps		Cloud Computing	Database & Analytics	Mobile Client
START TIME	END TIME			
8:00 AM	12:00 PM	M01 Workshop: DevOps in a Day - Brian Randell		
12:00 PM	2:00 PM			
2:00 PM	5:30 PM	M01 Workshop Continues		
7:00 PM	9:00 PM			
START TIME	END TIME			
8:30 AM	9:30 AM			
9:45 AM	11:00 AM	T01 Windows Presentation Foundation (WPF) 4.6 - Laurent Bugnion	T02 Angular 2 101 - Deborah Kurata	
11:15 AM	12:30 PM	T06 Windows 10—The Universal Application: One App To Rule Them All? - Laurent Bugnion	T07 TypeScript for C# Developers - Chris Klug	
12:30 PM	2:00 PM			
2:00 PM	3:15 PM	T11 Strike Up a Conversation with Cortana on Windows 10 - Walt Ritscher	T12 Angular 2 Forms and Validation - Deborah Kurata	
3:15 PM	3:45 PM			
3:45 PM	5:00 PM	T16 Building Truly Universal Applications - Laurent Bugnion	T17 Debugging the Web with Fiddler - Ido Flatow	
5:00 PM	6:30 PM			
START TIME	END TIME			
8:00 AM	9:15 AM	W01 Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - Benjamin Day	W02 Richer MVC Sites with Knockout JS - Miguel Castro	
9:30 AM	10:45 AM	W06 Team Foundation Server 2015: Must-Have Tools and Widgets - Richard Hundhausen	W07 Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - Benjamin Day	
11:00 AM	12:00 PM			
12:00 PM	1:30 PM			
1:30 PM	2:45 PM	W11 Stop the Waste and Get Out of (Technical) Debt - Richard Hundhausen	W12 Getting Started with Aurelia - Brian Noyes	
2:45 PM	3:15 PM			
3:15 PM	4:30 PM	W16 Real World VSTS Usage for the Enterprise - Jim Szubryt	W17 AngularJS & ASP.NET MVC Playing Nice - Miguel Castro	
6:45 PM	9:00 PM			
START TIME	END TIME			
8:00 AM	9:15 AM	TH01 Enterprise Reporting from VSTS - Jim Szubryt	★ TH02 Build Real-Time Websites and Apps with SignalR - Rachel Appel	
9:30 AM	10:45 AM	TH06 App Services Overview—Web, Mobile, API and Logic Oh My... - Mike Benkovich	TH07 Breaking Down Walls with Modern Identity - Eric D. Boyd	
11:00 AM	12:15 PM	TH11 Real-World Azure DevOps - Brian Randell	★ TH12 What You Need To Know About ASP.NET Core and MVC 6 - Rachel Appel	
12:15 PM	2:15 PM			
2:15 PM	3:30 PM	TH16 Cloud Oriented Programming - Vishwas Lele	TH17 Predicting the Future Using Azure Machine Learning - Eric D. Boyd	
3:45 PM	5:00 PM	TH21 Migrating Cloud Service Applications to Service Fabric - Vishwas Lele	★ TH22 Test Drive Automated GUI Testing with WebDriver - Rachel Appel	
START TIME	END TIME			
8:00 AM	5:00 PM	F01 Workshop: Data-Centric Single Page Apps with Angular 2, Breeze, and Web API - Brian Noyes		

Speakers and sessions subject to change.



CONNECT WITH VISUAL STUDIO LIVE!



twitter.com/vslive - @VSLive



facebook.com - Search "VSLive"



linkedin.com - Join the "Visual Studio Live" group!

# REDMOND AGENDA AT-A-GLANCE

Software Practices	Visual Studio / .NET Framework	Web Client	Web Server	Windows Client	Microsoft Exam Prep/ FREE Testing*
Visual Studio Live! Pre-Conference Workshops: Monday, August 8, 2016 <i>(Separate entry fee required)</i>					
M02 Workshop: Developer Dive into SQL Server 2016 - Leonard Label			M03 Workshop: Distributed Cross-Platform Application Architecture - Rockford Lhotka & Jason Bock		
Lunch @ The Mixer • Visit the Microsoft Company Store & Visitor Center					
M02 Workshop Continues			M03 Workshop Continues		
Dine-A-Round Dinner					
Visual Studio Live! Day 1: Tuesday, August 9, 2016					
★ KEYNOTE: To Be Announced, <i>Amanda Silver, Director of Program Management, Visual Studio, Microsoft</i>					
T03 Go Mobile with C#, Visual Studio, and Xamarin - James Montemagno		T04 What's New in SQL Server 2016 - Leonard Label		★ T05 Exam 70-480 Prep: Programming in HTML5 with JavaScript and CSS3*	
T08 Using Visual Studio Tools for Apache Cordova to Create Multi-Platform Applications - Kevin Ford		T09 No Schema, No Problem! – Introduction to Azure DocumentDB - Leonard Label		★ T10 Exam 70-483 Prep: Programming in C#*	
Lunch • Visit Exhibitors					
T13 Create Great Looking Android Applications Using Material Design - Kevin Ford		T14 Dependencies Demystified - Jason Bock		★ T15 .NET Core Cross Platform	
Sponsored Break • Visit Exhibitors					
★ T18 Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - Nick Landry		T19 Improving Performance in .NET Applications - Jason Bock		★ T20 ASP.NET Core	
Microsoft Ask the Experts & Exhibitor Reception • Attend Exhibitor Demos					
Visual Studio Live! Day 2: Wednesday, August 10, 2016					
W03 Windows for Makers: Raspberry Pi, Arduino & IoT - Nick Landry		W04 Applying S.O.L.I.D. Principles in .NET/C# - Chris Klug		W05 ASP.NET Core: Reimagining Web Application Development in .NET - Ido Flatow	
W08 Build Cross-Platform Mobile Apps with Ionic, Angular, and Cordova - Brian Noyes		W09 Open Source Software for Microsoft Developers - Rockford Lhotka		★ W10 Visual Studio vNext	
★ GENERAL SESSION: To Be Announced					
Birds-of-a-Feather Lunch • Visit Exhibitors					
W13 Top 10 Entity Framework Features Every Developer Should Know - Philip Japikse		★ W14 Creating Dynamic Pages Using MVVM and Knockout.JS - Christopher Harrison		★ W15 VSCode	
Sponsored Break • Exhibitor Raffle @ 2:55 pm (Must be present to win)					
W18 Take Your Site From Ugh to OOH with Bootstrap - Philip Japikse		W19 Debugging Your Way through .NET with Visual Studio 2015 - Ido Flatow		★ W20 Building A Connected Mobile App	
Set Sail! VSLive's Seattle Sunset Cruise on Lake Washington					
Visual Studio Live! Day 3: Thursday, August 11, 2016					
TH03 Write Better C# - James Montemagno		TH04 Pretty, Yet Powerful. How Data Visualization Transforms the Way we Comprehend Information - Walt Ritscher		★ TH05 Docker and Azure - Steve Lasker	
TH08 This session is sequestered, details will be released soon.		TH09 Windows 10 Design Guideline Essentials - Billy Hollis		★ TH10 Security and Authorization in .NET Core - Barry Dorrans	
TH13 Build Distributed Business Apps Using CSLA .NET - Rockford Lhotka		TH14 Learning to Live Without Data Grids in Windows 10 - Billy Hollis		★ TH15 Mobile Apps with JavaScript	
Lunch @ The Mixer • Visit the Microsoft Company Store & Visitor Center					
TH18 PowerShell for Developers - Brian Randell		TH19 Busy JavaScript Developer'sGuide to ECMAScript 6 - Ted Neward		★ TH20 Creating Windows Apps	
TH23 Tour d'Azure 2016 Edition...New Features and Capabilities that Dev's Need to Know - Mike Benkovich		TH24 Busy Developers Guide to Node.js - Ted Neward		★ TH25 TypeScript Deep Dive	
Visual Studio Live! Post-Conference Workshops: Friday, August 12, 2016 <i>(Separate entry fee required)</i>					
	F02 Workshop: Building Business Apps on the Universal Windows Platform - Billy Hollis				



★ Session features a Microsoft Speaker.

■ \*Check out all the exam prep and testing details at [vslive.com/redmond](http://vslive.com/redmond)

**VSLIVE.COM/MSDN**





# Visual Studio 2015 with .NET Core Tooling

.NET Core RC2 is finally here, and this time it is, in fact, a true “Release Candidate” rather than an RC1 Beta masquerading as a release candidate (if that, considering all the changes that happened after it came out). Much of the development focus and press surrounding .NET Core is, of course, about its cross-platform capabilities. This focus on supporting Linux and Mac OS X has led not only to a new .NET API, but also to an accompanying runtime and even a toolset. Tools such as DOTNET.EXE (formerly DNX, DNVM and DNU), not to mention Visual Studio Code, are all about giving non-Microsoft developers an opportunity to leverage .NET, and even more, a first-class developer experience.

This is all very well—and, in fact, amazing—but what about long-time Microsoft developers? How do they continue development on Windows and Visual Studio with .NET Core projects? In this article I’m going to talk about the various .NET Core project types and explain the details of the new file types and their functions. I’ll also delve into how the new project structure supports side-by-side debugging of any open source NuGet packages you might reference, and how you can step into the source code for such projects.

## Getting Started

Before you can begin leveraging .NET Core RC2, you’ll obviously want to update your tools—in this case Visual Studio 2015—to support the new platform. This essentially involves two steps:

1. Downloading the .NET Core Tooling Preview 1 for Visual Studio 2015 from [microsoft.com/net/core#windows](http://microsoft.com/net/core#windows) (where you’ll also find instructions).
2. Installing the NuGet Package Manager from [bit.ly/27Rmeaj](http://bit.ly/27Rmeaj).

I’m assuming you have Visual Studio 2015 installed already, but if not, Visual Studio Community Edition is available for free at [visualstudio.com](http://visualstudio.com).

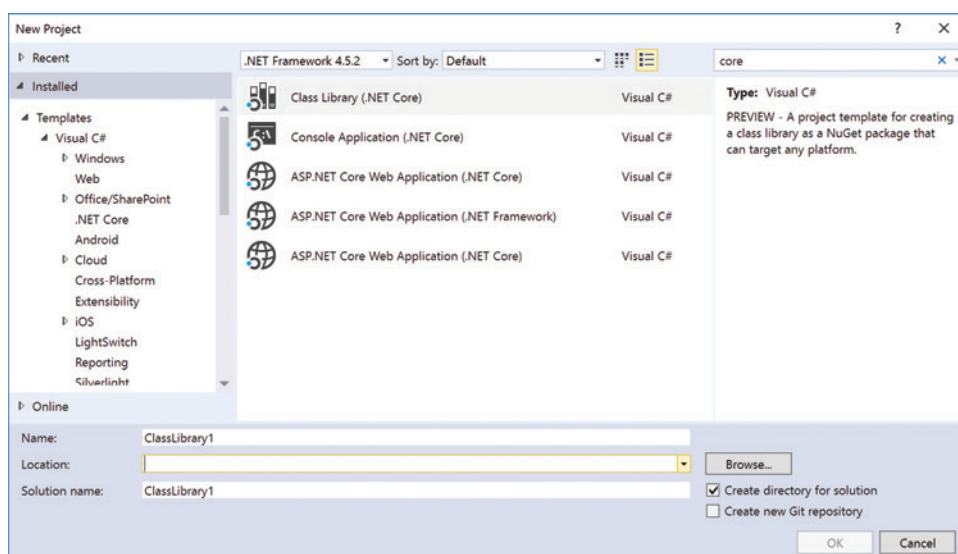


Figure 1 Visual Studio .NET Core Project Templates

## New Project Types

Once all the Visual Studio tooling is installed you can use the New Project wizard to create your project (see Figure 1).

As you can see, there are four project types (one appears twice as it resides in both the Visual C#\Web and Visual C#\\.NET Core folders).

Obviously, each project type generates a different set of files, as shown in Figure 2.

**App.config** is an optional configuration file similar to the traditional <appname>.config, used since the .NET Framework 1.0. The following code shows the default file, which identifies whether to use server-based garbage collection or client/workstation-type garbage collection (see [bit.ly/22nm32o](http://bit.ly/22nm32o) for more information):

```
<configuration>
  <runtime>
    <gcServer enabled="true"/>
  </runtime>
</configuration>
```

**AssemblyInfo.cs** identifies assembly data, such as configuration, company, product and trademark. This is the standard AssemblyInfo.cs file that has been part of Visual Studio .NET projects since .NET was first released in 2000.

**Class1.cs** is a skeleton C# class file for the Class1 class and includes its default constructor.

**Global.json** is generated automatically as long as you select “Create a directory for your solution” when creating your first .NET

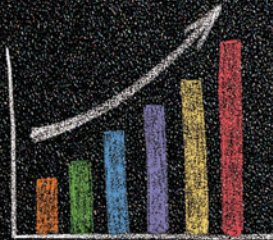


# Spreadsheets Made Easy.



## Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.



## Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.



Windows  
Forms



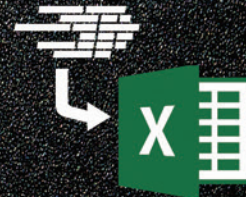
Silverlight



WPF

## Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.



## Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at [SpreadsheetGear.com](http://SpreadsheetGear.com)



# SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | [sales@spreadsheetgear.com](mailto:sales@spreadsheetgear.com)



Figure 2 The Files Included in Each Visual Studio Project Type

File Name	Class Library	Console Application	ASP.NET Core Web Application (.NET Core)	ASP.NET Core Web Application (.NET Framework)
App.config				X
Properties\AssemblyInfo.cs	X	X		
Class1.cs	X			
Global.json	X	X	X	X
Properties\launchSettings.json			X	X
Program.cs		X	X	X
Project.json	X	X	X	X
Project.lock.json	X	X	X	X
Project_Readme.html			X	X
<Project>.xproj			X	X
<Project>.xproj.user			X	X
Startup.cs			X	X
Web.config			X	X

Core project. As detailed later in the article, the project's node identifies additional locations for source code when debugging.

**LaunchSettings.json** identifies the various Web hosting configuration settings, including an application URL for debugging; an IIS host (such as IIS Express) if any; environment variables to set before launching, used, for example, by .NET Core Configuration to identify the environment (development, test or production); and SSL and authentication. Changes made from the Debug tab on Project Properties Window of Visual Studio provide a UI for editing the launchSettings.json file, as shown in **Figure 3**.

**Figure 4** shows a sample launchSettings.json file.

**Project.json** is a new project file whose functionality overlaps, in large part, with that of the \*.PROJ file. It identifies such things as project references, build options like the version number, and identifies the platform to compile to—whether .NET Core or .NET Framework, for example. More on this shortly.

**Project.lock.json** stores a list of the files (usually NuGet references) needed for compilation. It includes specific package version numbers, unlike the project.json file, which supports wildcards. Without project.lock.json, an entire restore of packages will occur. Project.lock.json includes a package graph along with other data related to packages that have been downloaded locally (restored). Generally, this file isn't checked in and, when it doesn't exist, it's recreated by running a NuGet package

restore. This file is listed as a child of project.json within Visual Studio.

**ClassLibrary.xproj** is the MSBuild file that, out of the box, defines what will happen when you build the project. The latest version imports Microsoft.DotNet.targets, which defines build tasks that leverage the new DotNet.exe command. Unlike MSBuild proj files in the past, xproj files are surprisingly small because much of the information has moved (for the moment) into project.json.

**ClassLibrary.xproj.user** overrides the ClassLibrary.xproj file and provides additional MSBuild properties such as local user debug-specific settings like ports. Generally, this file isn't checked in because it's user-specific, and it's not visible from within Visual Studio.

**Program.cs** defines a Program class that includes the definition of the Main entry point for your application—even for Web applications.

**Web.config** provides a minimal configuration for IIS, instructing it where to find the Web host process and configuring that all traffic is redirected to this

process, as shown in the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule"
        resourceType="Unspecified"/>
    </handlers>
    <aspNetCore processPath="%LAUNCHER_PATH%" arguments="%LAUNCHER_ARGS%"
      stdoutLogEnabled="false" stdoutLogFile=".\logs\stdout"
      forwardWindowsAuthToken="false"/>
    </system.webServer>
  </configuration>
```

Notice that web.config no longer contains app settings, which are instead moved into configuration files that are loaded by Microsoft.Extensions.Configuration as I outlined in my February 2016 article, "Configuration in .NET Core" ([bit.ly/10oqmkJ](http://bit.ly/10oqmkJ)).

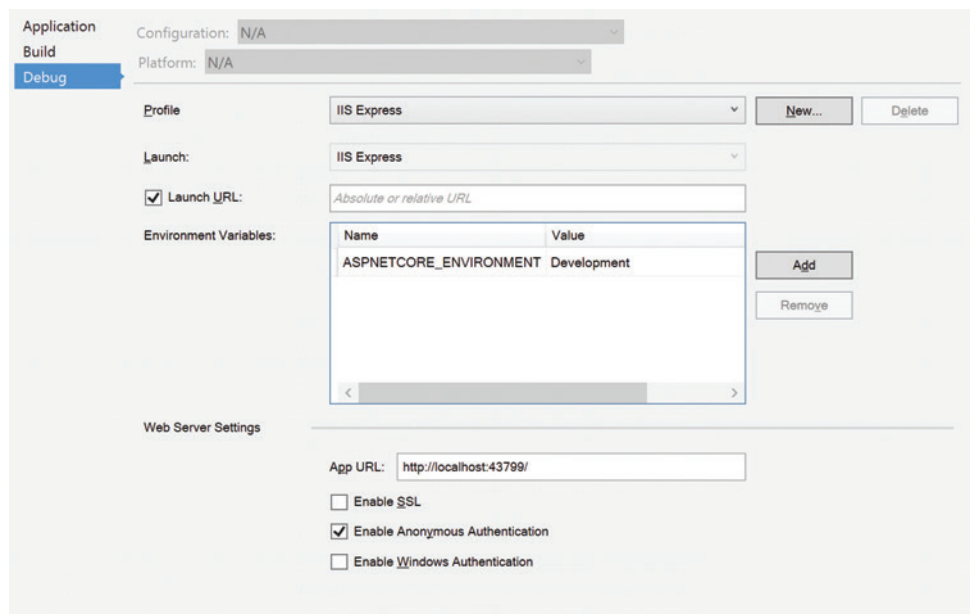


Figure 3 Debug Tab on the Project Properties Window of Visual Studio





**LIVE!**  
**360**

TECH EVENTS WITH PERSPECTIVE

**6**

*Great Conferences*

**1**

*Great Price*

# Orlando 2016

ROYAL PACIFIC RESORT AT UNIVERSAL  
DECEMBER 5-9

Visual Studio **LIVE!**

ModernApps **LIVE!**

**TECHMENTOR**

Office &  
SharePoint **LIVE!**

SQL Server **LIVE!**

**APPDEV** NEW!  
**TRENDS**

## *The Ultimate Education Destination*

**Live! 360<sup>SM</sup>** is a unique conference where the IT and Developer community converge to debate leading edge technologies and educate themselves on current ones. These six co-located events incorporate knowledge transfer and networking, along with expert education and training, as you create your own custom conference, mixing and matching sessions and workshops to best suit your needs.

**Choose the ultimate education destination: Live! 360.**



**REGISTER BY AUGUST 31  
AND SAVE \$500!**

Use promo code L3601  
Scan the QR code to register  
or for more event details.



**LIVE360EVENTS.COM**

EVENT PARTNERS



Microsoft

**Magenic**

PLATINUM SPONSOR



**GRIDSTORE**

GOLD SPONSOR



**GOVERLAN**  
COMPLEX IT SYSTEMS MANAGEMENT  
MADE EASY

SILVER  
SPONSOR



PRODUCED BY

**i105 MEDIA**

Figure 4 Sample launchSettings.json File

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:43799/",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "WebApplication1NetFramework": {
      "commandName": "Project",
      "launchBrowser": true,
      "launchUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Note that with all .NET Core project types created by Visual Studio, selecting the “Create a directory for your solution option” when creating a new project places all the source code into an `src` subdirectory for the solution. Furthermore, it’s expected that test projects will be placed into a test directory alongside `src` (although this will not happen by default in the .NET Core Tooling Preview 1 for Visual Studio 2015 release). Other possible directories to consider would be for things like builds and docs. (I confess I’d much prefer to keep my test projects alongside the target project they’re testing, rather than in an entirely separate tree, but this isn’t what Microsoft selected for the default solution structure, and experience tells me to embrace the default where possible rather than attempt to fight it.)

## More on Project.json

Perhaps the most significant file in the new .NET Core project structure is `project.json`. This file is designed to:

- Replace the NuGet File Manager `package.config` file, which identifies the NuGet references for the project.
- Specify which frameworks the project supports, and configuration details for how to build the project for the specific framework.
- Identify the target platform for self-contained apps, which contain all their dependencies including the platform-specific .NET Core runtimes needed for the corresponding platform. Or, if the project is a portable app, `project.json` identifies which frameworks the project expects to be installed on the target machine on which the assembly will execute.

These three tasks are spread across four main sections within `project.json` (I combine runtimes and supports as the functionality overlaps, depending on the project type):

**Dependencies:** This section lists each of the NuGet packages on which your project depends, including the version number of said dependencies. Version numbers can be specified with wild cards so you can allow “the most recent version” that matches the wild card to be downloaded automatically by the NuGet Package Manager restore. A pair of empty quotes for the version number says, “Use the latest available.” Furthermore, while you can add references via the Visual Studio NuGet Package Manager window, you can also rely on the way Visual Studio asynchronously loads IntelliSense with the packages available from your configured package source or feed (see **Figure 5**). The IntelliSense works for both the package name and the version number.

**Frameworks:** In this section you identify the frameworks on which your project will run: `net45`, `netstandard1.5`, `net40`, for example. In addition, you can provide `buildOptions`, `dependencies`, `frameworkAssemblies`, and `imports`—app-specific to the identified framework. For example, to allow unsafe code for the .NET 45 execution environment, you can specify within the frameworks element:

```
"net45": { "buildOptions": { "allowUnsafe": true } }
```

**Runtimes/supports:** Whether `project.json` uses *runtimes* or *supports* depends on whether the project is targeted as a portable or a self-contained app. For self-contained apps, the runtimes section identifies which OSes will be supported and, therefore, which runtime libraries to bundle into the application. A self-contained application has no dependencies (at least as far as .NET is concerned) on anything preinstalled on the target machine.

Whether `project.json` uses *runtimes* or *supports* depends on whether the project is targeted as a portable or a self-contained app.

In contrast, the `supports` section identifies for portable apps the runtime dependencies an app will look for when it starts up—one of which will be sufficient. The list isn’t restrictive (you may be able to run elsewhere), but a `supports` element will cause NuGet to check that all dependencies are satisfied.

While `project.json` is critical within RC1 and RC2, ironically, not long after RC2 was released, the .NET Core and ASP.NET teams

determined that `project.json` was, in fact, redundant to the already well-established MSBuild project file, which already supports significantly more functionality, including configuration, build dependencies, build targets, compilation flags, command-line and environment-variable property settings, and build commands. Rather than reinvent all this, the team reviewed what triggered the `project.json`

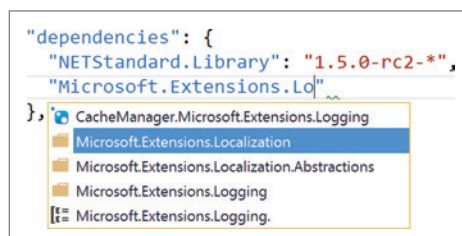


Figure 5 IntelliSense Dynamically Loads with the List of Available Packages and Versions



file creation in the first place and realized that \*.PROJ files were generally very bloated—listing every file within a project individually, for example, rather than using globbing patterns (wild cards). The bloat is so significant, in fact, that developers rarely open and review the file before running a build, even though potentially the file could have some undesirable commands embedded in it. Rather than creating an entirely new build file, the team instead decided to fix the bloat within \*.PROJ files and even provide a command-line utility for editing it. The only thing project.json has that \*.PROJ files don't is the JSON syntax. (But, after all, JSON is just the modern-day XML, just like XML was [in its day] the modern-day flat file or INI file.) For more information on project.json's potential demise, see Damian Edwards' announcement during the ASP.NET May 10, 2016, Community Standup and the corresponding post by Alexandre Mutel ([bit.ly/1NJ9r31](http://bit.ly/1NJ9r31)), along with Willem Meints summary on his blog ([bit.ly/1sJc2An](http://bit.ly/1sJc2An)).

## Debugging Package Source Code

One of the features I'm most excited about is the new support for debugging and stepping into packages, and even modifying the source code of the packages if it's available. Suppose, for example, you have a company-wide "Framework" assembly that's shared among numerous teams. However, the Framework package is essentially open source, such that anyone within the company (or, even better, outside the company) can contribute improvements and changes. Now, imagine if you reference the NuGet package for this Framework but at some point suspect there might be a bug that needs fixing or perhaps an enhancement is warranted. Normally, this requires working with the source code from the component independently of your project/solution. What if, instead, you were able to download the source code and update it as an integrated experience alongside your main development—even stepping into the code without relying on a symbol server or PDB files being available? Fortunately, this is a key scenario enabled in Visual Studio 2015.

One of the features I'm most excited about is the new support for debugging and stepping into packages, and even modifying the source code of the packages if it's available.

For example, imagine you want to debug the Microsoft.Extensions.Logging package, which is available on GitHub. To add and debug it within your project, you want to download (perhaps using the git clone or git submodule command) the source code. Next, in order for Visual Studio to know where to find the source code, you need to edit the global.json project node, for example, adding "submodules\Logging" to the list of directories it watches:

```
{
  "projects": [ "src", "test", "submodules\Logging" ],
  "sdk": {
    "version": "1.0.0-*"
  }
}
```

You can, of course, provide the full path (if, for example, you didn't clone the code into a subdirectory). However, note that the directory separator is either two back slashes (\\) or a single forward slash (for example, c:/users/mark/documents/visual studio2015/Projects/Microsoft.Extensions.Logging).

Rather than creating an entirely new build file, the team instead decided to fix the bloat within \*.PROJ files and even provide a command-line utility for editing it.

Once Visual Studio successfully finds the source code after global.json is updated and saved, it will automatically add the project to your solution such that you can then debug into the source code.

A fairly naïve algorithm is used to determine which source code directory to load:

1. If any of the source code locations specified in global.json contains a folder with the same name as the package (such as Microsoft.Extensions.Logging) and that folder contains a file named project.json, the debugger will use that folder and the source files inside it.
2. Otherwise, the compiled binary from the packages folder is loaded.

See "Debugging ASP.NET 5 Framework Code Using Visual Studio 2015" ([bit.ly/22njl7w](http://bit.ly/22njl7w)) for more information.

## Wrapping Up

If you haven't yet started to dive into .NET Core, now is a great time to do so, giving you the longest time span in which to amortize the learning curve. For those of you contemplating upgrading from earlier versions, the same is true. Chances are good you'll be upgrading at some point, and the sooner you do, the sooner you can take advantage of its new features. ■

---

**MARK MICHAELIS** is founder of IntelliTect, where he serves as its chief technical architect and trainer. For nearly two decades he has been a Microsoft MVP, and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books including his most recent, "Essential C# 6.0 (5th Edition)" ([itl.tc/EssentialCSharp](http://itl.tc/EssentialCSharp)). Contact him on Facebook at [facebook.com/Mark.Michaelis](https://facebook.com/Mark.Michaelis), on his blog at [IntelliTect.com/Mark](http://IntelliTect.com/Mark), on Twitter: @markmichaelis or via e-mail at [mark@IntelliTect.com](mailto:mark@IntelliTect.com).

---

**THANKS** to the following IntelliTect technical experts for reviewing this article: Kevin Bost, Andrew Scott and Michael Stokesbary





# Build a Wi-Fi Scanner in the UWP

Wi-Fi has over the last decade or so become ubiquitous. Many shops and cafes offer free Wi-Fi to customers for their convenience. Virtually all hotels offer some kind of wireless Internet to their guests. Most of us have wireless networks at home. As few tablets and mobile devices have Ethernet jacks, Wi-Fi has become integral to our modern lives. Beyond that, we rarely give it much thought.

So, questions abound. What about the sheer volume of Wi-Fi networks around us? How many are there? Are they secured? What channel are they on? What are they named? Can we map them? What can we learn from Wi-Fi network metadata?

While walking my dogs recently, I happened to glance at my phone's Wi-Fi network connection screen and noticed some witty network names. This got me to wonder how many others had chosen to be comical versus practical. Then, I had the idea to map out and scan wireless networks in and around my neighborhood. If I could automate the process, I could even scan and map wireless networks during my commute to work. Ideally, I could have a program running on a Raspberry Pi that would periodically scan wirelessly and record that data to a Web service. This certainly would be more practical than glancing at my phone intermittently.

As it turns out, the Universal Windows Platform (UWP) provides rich access to wireless network data via classes in the `Windows.Devices.WiFi` namespace. As you know, a UWP app can run not only on phones and PCs, but on Raspberry Pi 2 running Windows IoT Core. Now, I had all I needed to build out my project.

In this column, I'll explore the basics of scanning Wi-Fi networks using the APIs built right into the UWP.

## Windows.Devices.WiFi Namespace

The classes inside the `Windows.Devices.WiFi` namespace contain everything needed to scan and explore wireless adapters and wireless networks within range. After creating a new UWP project in Visual Studio, add a new class called `WifiScanner` and add the following property:

```
public WifiAdapter WifiAdapter { get; private set; }
```

Because it's possible to have multiple Wi-Fi adapters on a given system, you must pick the Wi-Fi adapter you want to use. The `InitializeFirstAdapter` method gets the first one enumerated in the system, as shown in **Figure 1**.

## Adding the Wi-Fi Capability

You might notice that there's a check for access to the Wi-Fi and that the code throws an exception if the `RequestAccessAsync` method returns

false. This is because the app needs to have a device capability to let it scan and connect to Wi-Fi networks. This capability isn't listed in the Capabilities tab in the manifest properties editor. To add this capability, right-click on the `Package.appxmanifest` file and choose View Code.

You'll now see the raw XML of the `Package.appxmanifest` file. Inside the Capabilities node, add the following code:

```
<DeviceCapability Name="wifiControl" />
```

Now save the file. Your app now has permission to access the Wi-Fi APIs.

## Exploring Wireless Networks

With the code to identify a Wi-Fi adapter to work with and permission to access it, the next step is to actually scan for networks. Fortunately, the code to do that is fairly simple; it's just a call to the `ScanAsync` method on the `WifiAdapter` object. Add the following method to the `WifiScanner` class:

```
public async Task ScanForNetworks()
{
    if (this.WifiAdapter != null)
    {
        await this.WifiAdapter.ScanAsync();
    }
}
```

Once `ScanAsync` runs, the `NetworkReport` property of the `WifiAdapter` gets populated. `NetworkReport` is an instance of `WifiNetworkReport`, which contains `AvailableNetworks`, a `List<WifiAvailableNetwork>`. The `WifiAvailableNetwork` object

**Figure 1 Find the First Wi-Fi Adapter Connected to the System and Initialize It**

```
private async Task InitializeFirstAdapter()
{
    var access = await WifiAdapter.RequestAccessAsync();

    if (access != WifiAccessStatus.Allowed)
    {
        throw new Exception("WifiAccessStatus not allowed");
    }
    else
    {
        var wifiAdapterResults =
            await DeviceInformation.FindAllAsync(WifiAdapter.GetDeviceSelector());

        if (wifiAdapterResults.Count >= 1)
        {
            this.WifiAdapter =
                await WifiAdapter.FromIdAsync(wifiAdapterResults[0].Id);
        }
        else
        {
            throw new Exception("Wifi Adapter not found.");
        }
    }
}
```

Code download available at [bit.ly/1UrY3rz](http://bit.ly/1UrY3rz).

Figure 2 The XAML for the UI

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition Height="60"/>
    <RowDefinition Height="60"/>
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <TextBlock FontSize="36" Grid.RowSpan="2" >WiFi Scanner</TextBlock>
  <StackPanel Name="spButtons" Grid.Row="1" Orientation="Horizontal">
    <Button Name="btnScan" Click="btnScan_Click" Grid.Row="1">Scan For
      Networks</Button>
  </StackPanel>
  <TextBox Name="txtReport" TextWrapping="Wrap" AcceptsReturn="True"
    Grid.Row="2"></TextBox>
</Grid>
</Page>
```

contains numerous data points about a given network. You can find the Service Set Identifier (SSID), signal strength, encryption method and access point uptime, among other data points, all without connecting to the network.

Iterating through the available networks is quite easy: You create a Plain Old CLR Object (POCO) to contain some of the data from the `WiFiAvailableNetwork` objects, as seen in the following code:

```
foreach (var availableNetwork in report.AvailableNetworks)
{
    WiFiSignal wifiSignal = new WiFiSignal()
    {
        MacAddress = availableNetwork.Bssid,
        Ssid = availableNetwork.Ssid,
        SignalBars = availableNetwork.SignalBars,
        ChannelCenterFrequencyInKilohertz =
            availableNetwork.ChannelCenterFrequencyInKilohertz,
        NetworkKind = availableNetwork.NetworkKind.ToString(),
        PhysicalKind = availableNetwork.PhyKind.ToString()
    };
}
```

## Building the UI

While I intend for the app to run without a UI in the final project, it's useful for development and troubleshooting to see the networks within range and the metadata associated with them. It's also useful for developers who might not have a Raspberry Pi at the moment, but still want to follow along. As shown in **Figure 2**, the XAML for the project is straightforward and there's a multiline `TextBox` to store the output of the scan.

## Capturing Location Data

In order to provide additional value, each scan of the wireless network should also note the location of the scan. This will make it possible to provide interesting insights and data visualizations

Figure 3 Code to Iterate Through All the Networks Found During a Scan

```
foreach (var availableNetwork in report.AvailableNetworks)
{
    WiFiSignal wifiSignal = new WiFiSignal()
    {
        MacAddress = availableNetwork.Bssid,
        Ssid = availableNetwork.Ssid,
        SignalBars = availableNetwork.SignalBars,
        NetworkKind = availableNetwork.NetworkKind.ToString(),
        PhysicalKind = availableNetwork.PhyKind.ToString(),
        Encryption = availableNetwork.SecuritySettings.NetworkEncryptionType.ToString()
    };

    wifiPoint.WiFiSignals.Add(wifiSignal);
}
```

later. Fortunately, adding location to UWP apps is simple. You will, however, need to add the Location capability to your app. You can do that by double-clicking on the `Package.appxmanifest` file in Solution Explorer, clicking on the Capabilities tab, and checking the Location checkbox in the Capabilities list.

The following code will retrieve the location using the APIs built into the UWP:

```
Geolocator geolocator = new Geolocator();
Geoposition position = await geolocator.GetGeopositionAsync();
```

Now that you have a location, you'll want to store the location data. Following is the `WiFiPointData` class, which stores location data along with information about networks found at the location:

```
public class WiFiPointData
{
    public DateTimeOffset TimeStamp { get; set; }
    public double Latitude { get; set; }
    public double Longitude { get; set; }
    public double Accuracy { get; set; }
    public List<WiFiSignal> WiFiSignals { get; set; }
    public WiFiPointData()
    {
        this.WiFiSignals = new List<WiFiSignal>();
    }
}
```

At this time, it's important to note that unless your device has a GPS device, then the app requires a Wi-Fi connection to the Internet in order to resolve location. Without an onboard GPS sensor, you'll have to have a mobile hotspot and make sure that your laptop or Raspberry Pi 2 is connected to it. This also means that the reported location will be less accurate. For more information on best practices for creating location-aware UWP, please refer to the Windows Dev Center article, "Guidelines for Location-Aware Apps," at [bit.ly/1P0St0C](http://bit.ly/1P0St0C).

## Scanning Repeatedly

For a scanning-and-mapping-while-driving scenario, the app needs to periodically scan for Wi-Fi networks. To accomplish this, you'll need to use a `DispatcherTimer` to scan for Wi-Fi networks at regular intervals. If you're not familiar with how `DispatcherTimer` works, please refer to the documentation at [bit.ly/1WPMFcp](http://bit.ly/1WPMFcp).

It's important to note that a Wi-Fi scan can take up to several seconds, depending on your system. The following code sets up a `DispatcherTimer` to fire an event every 10 seconds, more than enough time for even the slowest system:

```
DispatcherTimer timer = new DispatcherTimer();
timer.Interval = new TimeSpan(0, 0, 10);
timer.Tick += Timer_Tick;
timer.Start();
```

Every 10 seconds, the timer will run the code in the `Timer_Tick` method. The following code scans for Wi-Fi networks and then appends the results to the `TextBox` in the UI:

```
private async void Timer_Tick(object sender, object e)
{
    StringBuilder networkInfo = await RunWifiScan();
    this.txtReport.Text = this.txtReport.Text + networkInfo.ToString();
}
```

## Reporting Scan Results

As mentioned previously, once the `ScanAsync` method is called, the results of the scan are stored in a `List<WiFiAvailableNetwork>`. All it takes to get to those results is to iterate through the list. The code in **Figure 3** does just that and places the results into an instance of the `WiFiPointData` class.

Figure 4 Converting WiFiPointData

```
private StringBuilder CreateCsvReport(WiFiPointData wifiPoint)
{
    StringBuilder networkInfo = new StringBuilder();
    networkInfo.AppendLine("MAC,SSID,SignalBars,Type,Lat,Long,Accuracy,Encryption");

    foreach (var wifiSignal in wifiPoint.WiFiSignals)
    {
        networkInfo.Append($"({wifiSignal.MacAddress},");
        networkInfo.Append($"({wifiSignal.Ssid},");
        networkInfo.Append($"({wifiSignal.SignalBars},");
        networkInfo.Append($"({wifiSignal.NetworkKind},");
        networkInfo.Append($"({wifiPoint.Latitude},");
        networkInfo.Append($"({wifiPoint.Longitude},");
        networkInfo.Append($"({wifiPoint.Accuracy},");
        networkInfo.Append($"({wifiSignal.Encryption})");
        networkInfo.AppendLine();
    }

    return networkInfo;
}
```

In order to make the UI simple while still providing for rich data analysis, you can convert the WiFiPointData to a Comma Separated Value (CSV) format and set the text of the TextBox in the UI. CSV is a relatively simple format that can be imported into Excel and Power BI for analysis. The code to convert WiFiPointData is shown in Figure 4.

## Visualizing the Data

Naturally, I couldn't wait to set up my cloud service to display and visualize the data. Accordingly, I took the CSV data generated by the app and copied and pasted that into a text file. I then made sure to save the file with a .CSV extension. Next, I imported the data into Power BI Desktop. Power BI Desktop is a free download from [powerbi.microsoft.com](http://powerbi.microsoft.com) that makes it easy to visualize and explore data.

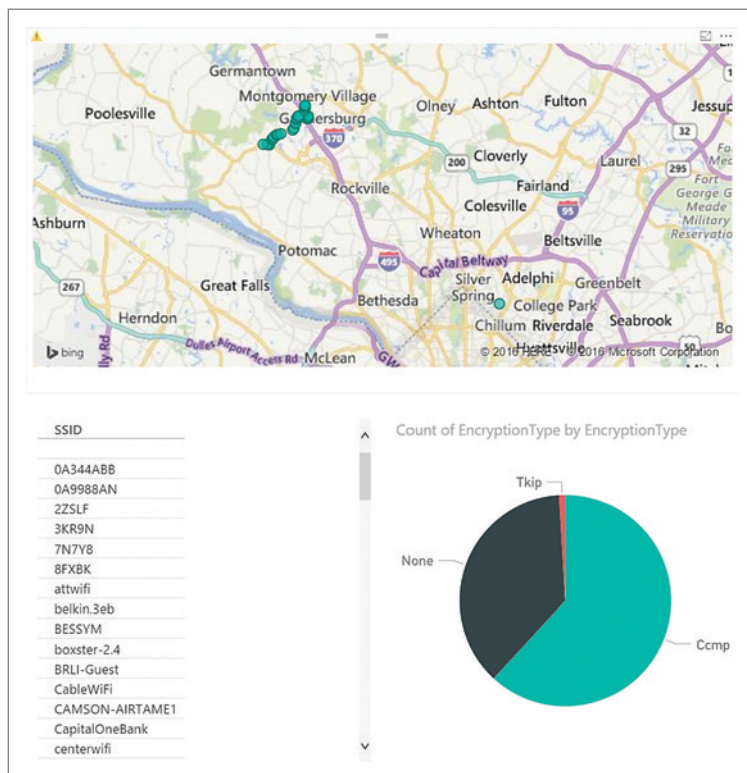


Figure 5 Power BI Visualization of Data the Wi-Fi Scanner App Collected

To import the data from the app, click on Get Data on the Power BI Desktop splash screen.

On the following screen, choose CSV and then click Connect. In the file picker dialog, choose the CSV file with the data copied and pasted out of the app.

Once that loads, you'll then see a list of fields on the right hand side of the screen. While a full tutorial on Power BI Desktop is beyond the scope of this article, it doesn't take much skill to produce a visualization that shows the location of Wi-Fi networks, their SSIDs and the encryption protocols they employ, as shown in Figure 5.

Amazingly, about one-third of networks are completely unencrypted. While some of these are guest networks set up at various businesses, some are not.

## Practical Applications

While the original intent was merely to measure the technical savvy and wit of my neighbors, this project has some rather interesting practical uses. The ability to easily and automatically map Wi-Fi signal strength and location has interesting applications. What could a city do if each city bus were outfitted with an IoT device with this app running on it? Cities could measure the prevalence of Wi-Fi networks and correlate that data with neighborhood income data. Elected officials could then make informed policy decisions based on that data. If a community provides public Wi-Fi across town or in certain areas, then the signal strength could be measured in real time without the added cost of sending technicians around. Cities could also determine where unsecured networks were prevalent and create targeted awareness programs to increase community cyber security.

On a smaller scale, the ability to quickly scan Wi-Fi network metadata comes in handy when setting up your own network. Many routers offer users the chance to modify the channel on which they broadcast. A great example of this is an app called "Wi-Fi Analyzer" ([bit.ly/25vZ0Q0](http://bit.ly/25vZ0Q0)), which, among other things, displays the strength and frequency of nearby wireless networks. This comes in handy when setting up a Wi-Fi network in a new location.

## Wrapping Up

Copying and pasting text data from the UI will not scale. Furthermore, if the goal is to run the app on an IoT device without any type of display, then the app needs to send data to the cloud without any UI. In my next month's column, you'll learn how to set up a cloud service to take in all this data. Additionally, you'll learn how to deploy the solution to a Raspberry Pi 2 running Windows IoT Core.

**FRANK LA VIGNE** is a technology evangelist on the Microsoft Technology and Civic Engagement team, where he helps users leverage technology in order to create a better community. He blogs regularly at [FranksWorld.com](http://FranksWorld.com) and has a YouTube channel called [Franks World TV](http://FranksWorldTV) ([youtube.com/FranksWorldTV](http://youtube.com/FranksWorldTV)).

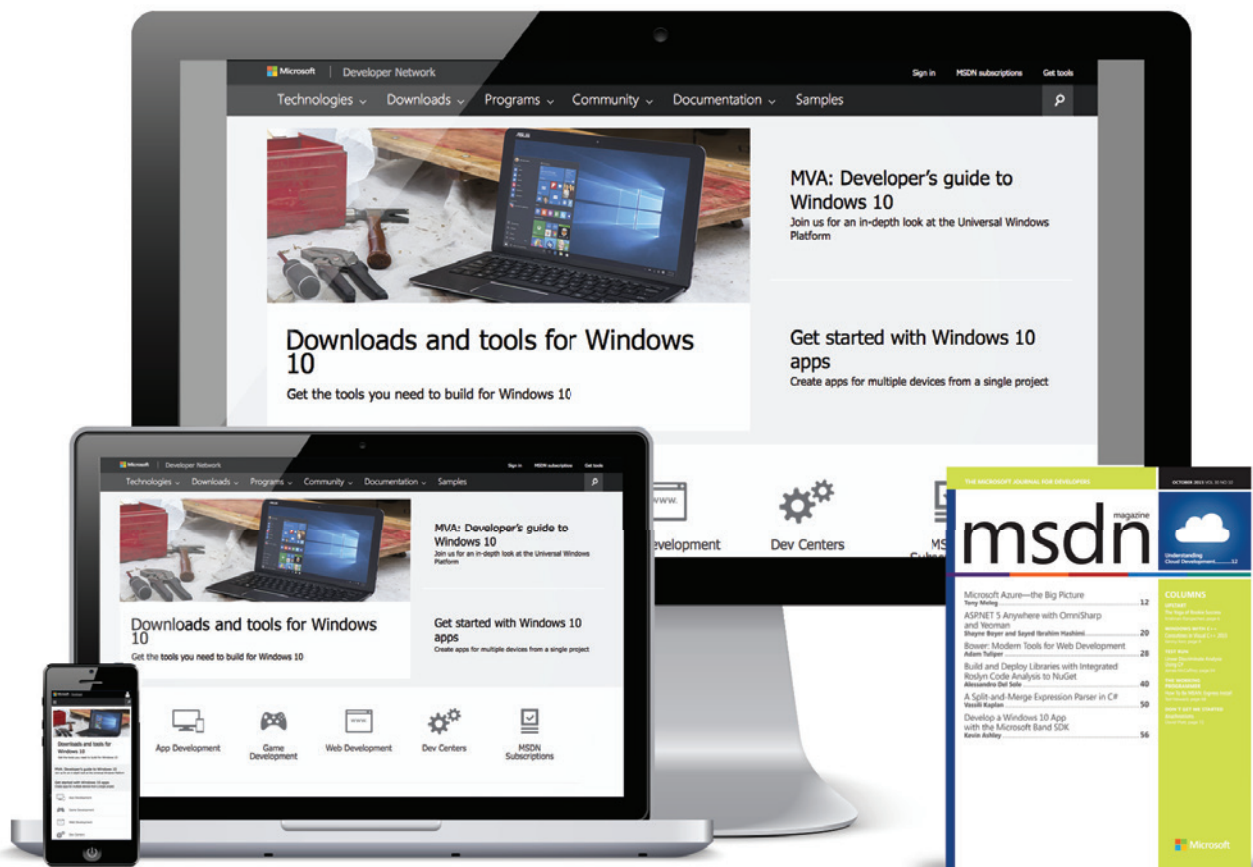
**THANKS** to the following technical experts for reviewing this article: Rachel Appel, Robert Bernstein and Jose Luis Manners



# msdn

magazine

Where you need us most.



Visual Studio **LIVE!**  
EXPERT SOLUTIONS FOR .NET DEVELOPERS

**LIVE!**  
**360**  
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com



# Missing the Target

I wrote several years ago about the importance of UX in the enterprise sector (“The Peasants Are Revolting,” March 2014, [msdn.com/magazine/dn605882](http://msdn.com/magazine/dn605882)). Beauty-products company Avon had to euthanize a \$125 million order system after its independent sales reps quit the company rather than use its terrible UX. But Avon is not alone. I’ve run across two new examples of bad UX in enterprise applications. One killed a company’s expansion plans, and the other may soon end up killing humans.

Target, the upscale downscale department stores popular in the United States, attempted to expand into Canada in 2013. The idea seemed promising, but Target couldn’t make it work. Joe Castaldo’s story in *Canadian Business* magazine describes its demise in detail ([bit.ly/1P88uj0](http://bit.ly/1P88uj0)): “a sophisticated retail giant felled by the most mundane, basic and embarrassing of errors.” I call it bad UX.

Modern retailers live and die by the data on which they manage their supply chain. Target’s existing software couldn’t handle international requirements, such as Canadian dollars and French-language characters. So Target decided to build a new implementation with SAP. (I’ve heard SAP described as “like peeling an onion—it has multiple layers and makes you want to cry.” Avon’s failure also happened in SAP. Coincidence? You tell me.) According to Castaldo, the data in the Canadian operation was accurate only about 30 percent of the time, as opposed to 98 percent to 99 percent in the United States. How the [expletive] did this happen?

Bad UX, simple as that. Human users had to enter information into SAP for each item the store carried. Each product could require dozens of fields: manufacturer, model, UPC, dimensions and so on. Being human, the users made entry errors, which the software didn’t catch.

I can imagine Target’s developers choosing to ignore UX. “It’s a data entry system, for heaven’s sake. We don’t need no stinking decoration.” At the very least, they needed to give some thought about how their users actually did work. For example, the software couldn’t reject a UPC code with a missing digit until more than a year into the project. Too little, too late.

Target abandoned its Canadian expansion effort in 2015, writing off about \$5 billion. With better UX, specifically for data entry, the company might still be there, eh?

*Hakuna matata*, you say. If Target isn’t open, I’ll just go down the street to Walmart. But it’s different when bad UX hobbles your doctor’s medical practice. Which is happening in Boston’s top hospitals as I write these words.

*The Boston Globe* ran a story on May 17, headlined: “New \$1.2b Partners Computer System Brings Prescription for Frustration.” (See [bit.ly/24XZfrE](http://bit.ly/24XZfrE); Partners is a local alliance of major teaching hospitals.)

The article reports users saying “it has an insatiable demand for information that, keystroke by keystroke, click by click, overwhelms the already tightly wrapped day inside a hospital, eats away at time with patients, and sometimes forces them to work longer shifts. Simple tasks like ordering medications and tests can take several minutes longer, forcing patients to wait around while staff navigate the system.”

I’ve heard SAP described as  
“like peeling an onion—it has  
multiple layers and makes  
you want to cry.”

One doctor reported that she spent half her days on the computer, with her back to her patients, to the point where “it usurp[ed] the physician-patient relationship.” She decided to retire early—shades of Avon. One maternity nurse says that she became “a captive of the keyboard, spending far more of her time recording every blood pressure reading, every feeding, every diaper change.” More than once she has burst into tears on the drive home.

The vendor says, “[Our] goal is to make the software a joy to use.” Unless the nurse is sobbing with happiness, it has a long way to go.

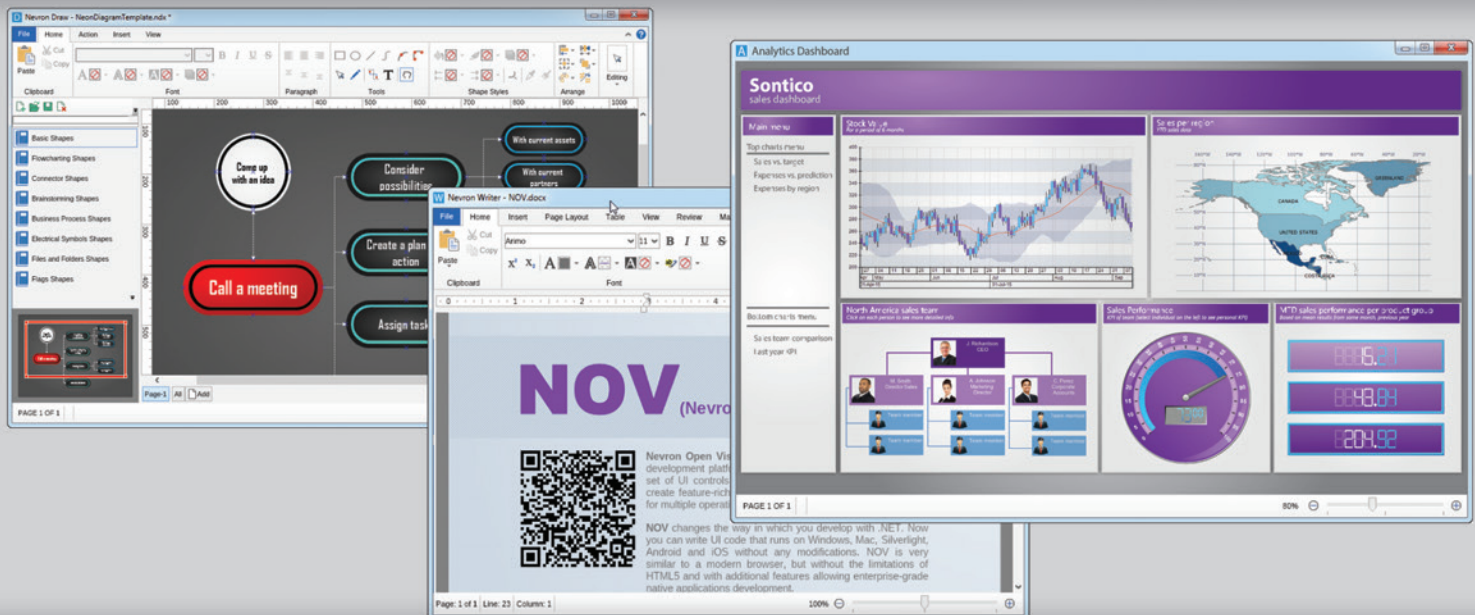
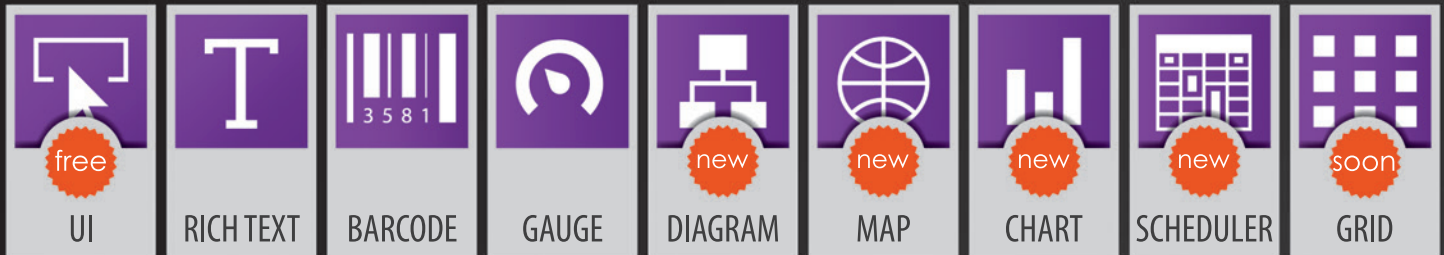
Dr. Gregg Meyer, chief clinical officer at Partners, “acknowledged the program includes more clicks that can slow doctors down. But he said the additional time is justified if it leads to safer care.”

I have a hard time seeing slower usage as a design goal. If it was, why not remove half the RAM from the computers, or make users wear mittens while typing? Or maybe start performing tonsillectomies through the rectum? That’ll slow them down, all right.

I tremble when I see a system piling more work onto mission-critical users who are already way overstretched. Why doesn’t it target less user effort, rather than more? Does a patient have to die before software vendors learn to put users first? ■

**DAVID S. PLATT** teaches programming .NET at Harvard University Extension School and at companies all over the world. He’s the author of 12 programming books, including “Why Software Sucks” (Addison-Wesley Professional, 2006) and “Introducing Microsoft .NET” (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter’s fingers so she learns how to count in octal. You can contact him at [rollthunder.com](http://rollthunder.com).

# The Complete UI suite for .NET



Nevron Open Vision is the first and only .NET component suite that allows you to develop cutting edge Applications, Dashboards and User Interfaces for Windows (WinForms and WPF) and Mac (MonoMac and Xamarin.Mac) from a Single Code Base.

It includes advanced UI controls such as Chart, Diagram, Grid, Text Editor, Scheduler, Gauges and Barcodes as well a complete set of UI controls (ListBox, TreeView, ComboBox, Color Pickers etc.).

The controls in those suites seamlessly integrate in

Microsoft Silverlight |  **WPF** |  **WinForms** |  **Xamarin**

All controls in the suite are completely windowless, styleable and open for customization.

Learn more at [www.nevron.com](http://www.nevron.com) today

[www.nevron.com](http://www.nevron.com) | [email@nevron.com](mailto:email@nevron.com) | +1 888-201-6088 (Toll free, USA and Canada)

Microsoft, .NET, ASP.NET, SharePoint, SQL Server and Visual Studio are registered trademarks of Microsoft Corporation in the United States and/or other countries. Some Nevron components are only available for certain platforms. For details visit [www.nevron.com](http://www.nevron.com) or send an e-mail to [support@nevron.com](mailto:support@nevron.com).



# SYNCFUSION SUCCINCTLY SERIES



90 titles and growing | Ad-free | 100 pages | PDF & Kindle formats

Download these books and more at: [syncfusion.com/msdnagile](https://syncfusion.com/msdnagile)

